TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALEXEY PATRUSHEV

ANALYSIS OF SOFTWARE ENGINEERING DATA WITH SELF-ORGANIZING MAPS

Master of Science thesis

Examiner: prof. Kari Systä
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 4th May 2016

ABSTRACT

Alexey Patrushev: Analysis of software engineering data with self-organizing maps, Tampere University of Technology
Master of Science Thesis, 72 pages, 17 Appendix pages
January 2016
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Professor Kari Systä
Keywords: Self-Organizing Maps, Software Engineering Data, Machine Learning

Nowadays software developers have tools that help to assist them during development and management processes of software products. These tools store large amount of Software Engineering data resulted from these processes. Analysis of the data can reveal valuable information about project performance and discover useful business patterns used during development. This information can be utilized to find projects where teams have some management problems and help to improve situation. Currently existing methods in the field have applicability limitations, because they require an expert knowledge for evaluation and are not capable to deal with large number of projects. In this thesis, we will explore possibility to apply Machine Learning methods to analysis of software engineering data. Machine Learning methods are capable to build a model from sample inputs and produce data-driven predictions. They have been gaining popularity over the last decades and show promising results in applications, where human expertise was traditionally used.

In this work, we attempt to extract and analyze software project management patterns from software engineering data stored in the GitHub repositories. For this purpose, we have developed a system, which is capable of collecting the project data, extracting their features and comparing properties of large number of projects between each other. To collect projects, we used Unified Data Model that is capable of storing of software engineering data from various sources; we have also spotted a few limitations of this model and have improved it to meet requirements of our work. Obtained data was used for training of Self-Organizing Maps. The resulted map have demonstrated clear grouping principles of the projects according to chosen feature set. We estimated efficiencies for distinct areas of the map. Effects of different events occurred during issue lifetime, such as user assignation and labeling, were also investigated. Based on data analysis, we showed that labeling and user assignation is beneficial and can potentially decrease issue resolution time. The main result of our work is evaluation system that is capable of data collection, storage, cleaning and evaluation. Evaluation part of our system was based on analysis of 230 individual projects that was result of cooperation of 100 000 unique users from GitHub community. Further research directions can include verification of estimation subsystem by GitHub users who participated in project development.

PREFACE

This work is ultimately based on data that is widely available online as well as product of comprehensive analysis performed on open-source projects data. The work is original and independent product written by the author in the Tampere University of Technology (TUT).

I am thankful for all who helped me with this work to my mentor for this amazingly interesting topic, to people who gave me feedback and answered my questions and for those who indirectly provided the data that was used in my work as result of parsing through the GitHub repositories. At the end I want to express gratitude to my family and friends, who helped me to carry out this research.

Tampere, 04.10.2016

Alexey Patrushev

CONTENTS

1. INTRODUCTION 1
1.1 Software project management 1
1.2 Possibility of analysis of the engineering data and related works 3
1.3 Objectives and main results of the thesis 4
1.4 Organization of the thesis 5
2. BACKGROUND KNOWLEDGE 6
2.1 Machine learning 6
2.2 Sample and its features 6
2.3 Learning Paradigms 7
2.3.1 The supervised learning 7
2.3.2 The unsupervised learning 7
2.4 Regression analysis problems 7
2.5 Classification problems 8
2.6 Cluster analysis problems 8
2.7 Dimension reduction problems 9
2.8 Artificial Neural Networks 10
2.8.1 The single neuron 10
2.8.2 Multilayer feed-forward neural networks 13
2.9 Competitive learning 14
2.10 The Self-Organizing Maps 15
2.10.1 Overview 15
2.10.2 The SOM algorithm 16
2.10.3 Competitive process 17
2.10.4 Cooperative Process 18
2.10.5 Adaptive Process 19
2.10.6 Practical advices on how to construct a good SOM 19
2.10.7 Properties of the Self-Organized Maps 21
2.11 Summary 22
3. METHODOLOGY 23
3.1 The Git and GitHub 23
3.1.1 Introduction to the GitHub 23
3.1.2 Brief outline of GitHub work-flow 24
3.1.3 GitHub events 25
3.2 Data collection 26
3.2.1 The Unified Data Model 26
3.2.2 Analysis of the Unified Model and its implementation 28
3.2.3 The modified model 31
3.3 The data collection step 37
3.4 Summary of obtained software engineering data from the GitHub 38

3.5 The data transformation and preparation....39
3.5.1 The four issue categories....39
3.5.2 The analysis of issue work-flow....42
3.6 The feature extraction....43
3.6.1 The statistical quantities used in feature extraction....43
3.6.2 Description of extracted features in relation to categories....44
3.7 The feature selection....46
3.8 Data cleaning....48
3.9 Data normalization....48
3.10 Summary....49
4. THE DATA ANALYSIS....50
4.1 The hypotheses....50
4.1.1 Hypotheses formulation....50
4.1.2 Evaluation of the hypotheses from statistical data....50
4.2 The SOM training....53
4.3 The resulting SOM and analysis of different areas, present in the map....53
4.4 The characteristics of the areas....55
4.4.1 The analysis of category related features....55
4.4.2 The analysis of supplementary features....63
4.4.3 The summary of results....65
4.5 The projects quality, intermediate results....66
5. EVALUATION....68
6. CONCLUSIONS AND FUTURE WORK....70
APPENDIX A: Database table creation
APPENDIX B: Projects collected from GitHub
APPENDIX C: TMPTBFILLING stored procedure
APPENDIX D: EVENTPARSER stored procedure
APPENDIX E: COMMITSPERUSER stored procedure

LIST OF FIGURES

Figure 1: Sample set of written symbols, produced by different individuals.....4

Figure 2: a) feature selection, b) feature extraction.....7

Figure 3: The sigmoid threshold unit.....8

Figure 4: Activation functions.....9

Figure 5: Examples of a) linearly separable and b) linearly non-separable sets. Positive examples are indicated by “+” and negative “-”.....10

Figure 6: Feed forward artificial neural network with three layers of neurons.....11

Figure 7: Kohonen map or self-organizing map, as preferred by Teuvo Kohonen [24]..13

Figure 8: Gaussian neighbourhood function.....16

Figure 9: BMU's and its local neighbourhood.....16

Figure 10: a) hexagonal and b) rectangular grids of neurons.....18

Figure 11: Learned semantic categories in a SOM model [27].....20

Figure 12: The Unified data model for software engineering data [28].....24

Figure 13: The architecture of program model for data collection, conversion, linking, sorting and visualization.....26

Figure 14: The implementation of database structure, acquired from script description for database creation and analysis of source codes of parser\linker program.....27

Figure 15: The Modified Model.....30

Figure 16: The architecture model.....36

Figure 17: TMPTBFILLING stored procedure used for division of all labels in four distinct categories.....43

Figure 18: The self-organizing map sample hits received in the output model.....53

Figure 19: The outline scheme and enumeration for nine different areas that were analyzed.....54

Figure 20: Intermediate region performance scores outline on the left side and corresponding performance measures, showed on the right side.....65

LIST OF SYMBOLS AND ABBREVIATIONS

AAN	Artificial neural network
API	Application Programming Interface
BMU	Best Matching Unit
BP	Back-Propagation
DB	Data-Base
DDL	Data Definition Language
DML	Data Manipulation Language
GUI	Graphical User Interface
ID	Identification
JS	Java Script
JSON	JavaScript Object Notation
ODBC	Open Data-Base Connectivity
OS	Operating System
OSS	Open Source Software
PCA	Principal Component Analysis
PSQL	Procedural Structured Query Language
RAD	Rapid Application Development
RDBMS	Relational Database Management System
Repo	Repository
SOM	Self-Organizing Maps
SPI	Software Process Improvement
SQL	Structured Query Language
STD	STandard Deviation
TUT	Tampere University of Technology
UDF	User-Defined Function
UI	User Interface
UML	Unified Modeling Language

1. INTRODUCTION

Today our world is interconnected, people can communicate with each other easier and almost from everywhere. This changed many aspects of our reality. For example, we can have friends from another side of the globe and can easier maintain our relationships. Similarly, interconnection opened a door for international cooperation in such fields as distributed development, multinational cooperation between corporations and etc. The distributed development generates a lot of data that are available online. This data can be analyzed in order to improve work-team efficiency and clarify status of the project.

In this work, we analyze software engineering data collected from projects that are freely available online. The data can be collected from various systems used for distributed development and version control purposes. These kinds of systems gained widespread application across many software engineering projects, with developers all over the world, and especially, in the open-source community. The data can be analyzed using machine learning methods revealing the best practices inside teams. With this information, software project management can be adjusted improving development process and user satisfaction scores.

1.1 Software project management

Software project management is difficult and truly complex task, because many factors are involved in the development process of a new software product. The quality of the developed software product relies on many different aspects starting from programmers personal skills to project managers competence and aspiration to design and deliver better product [2]. However, it is hard to analyze development process and find the reasons, why some particular project came to failure, exceeded budget, or did not meet the requirements. Comparison between succeeded and failed project is also a challenge, because there are many steps of analysis, which might be involved in the process:

1. Analysis of all documents and other information related to both projects (if this data will be available at the time) and/or performing expensive and time-consuming interviews.
2. Finding issues and generalizing them in the report.
3. Proposing some changes that can improve development process.

The changes can modify business process in a good way and allow a team to understand hidden pitfalls, improve quality of work and cooperation inside the group. Many teams, which just started a project, do not have access to service that can determine troublesome factors in the work-flow due to many reasons. These reasons could include the lack of the budget resources, the plan to deliver free product for the users, or the insufficient amount of money at the initial stages of the project. This is often the case for the new Open Source (OS) projects in early stages of development. Sometimes, they do not have devoted managers, who can direct the project and give new goals, which team will adopt and follow. In such OS projects, developers establish and later execute own goals based on own needs and project vision. This might lead to further complications of development process, creation of features that will not be frequently used by most of the users, or putting the focus on development of new features instead of fixing important bug issues. In such a way, development of Open Source project can end up in complicated user unfriendly software product [3].

Nowadays, many software projects can employ Open Source model, making the final product free of charge for users. Developers of such products earn money on professional support subscriptions. Open Source Software is an important software paradigm, where developers participate voluntary; sometimes, they are not paid for their work, but are committed to such hobby in their own free time. Nevertheless, such software systems maintain high functionality and quality that attract many users as free and affordable alternatives to proprietary solutions [4]. In addition, such projects allow developers with different background to interact with each other and to grow in professional manner. For example, one team can include mature senior software developers as well as students. The latter ones are challenged to get paid jobs, because of lack of experience, but are eager to work and want to acquire new skills. From this perspective, such cooperation seems even more important allowing for many people to cooperate in the team and to improve own skills. This kind of cooperation can be between the people who live in different countries, and team will be broadly distributed across the Globe.

Today, many systems coexist helping to connect many different members of one team and to accomplish the projects. This kind of systems usually provide version control features in addition to the platform for interaction between distributed developers. Some of them even have possibilities to include in this cooperation the users of end product. These systems include Git (a version control system), its expanded web-based version GitHub [5] (in addition, it includes several collaboration and issue management features), and Jira [6] (a proprietary issue tracking and project management system). Both of them have subscription fees for commercial and free access for open source projects. The GitHub system is easier to use for open source projects; it provides platform that allows an easy exchange and install options for a new user. Also, obtaining

free open source subscription with the Jira is slightly tricky, because this kind of projects need to provide their status and be approved through online registration form, which can take up to two weeks [7]. These factors contributed to widespread GitHub usage between many open source projects.

1.2 Possibility of analysis of the engineering data and related works

The engineering data, according to [8] is:

> Any information that collectively becomes the knowledge on which an engineer can design and build the proposed end-product. This information comes as drawings, manufacturer's specifications, and standards. Coupled with information relating to design, procurement, fabrication, test, and inspection of an item or structure, this rounds out the information from which the engineer designs and builds.

In relation to the GitHub, the engineering data is the information obtained during the development process. For instance, this information might include issues related to tasks, bugs, documents or user questions, events occurred and associated with these issues, commits, and source code written by project team. Basically, all kind of information that is kept on the GitHub can be considered as engineering data, because the creation and supporting phases of output product were influenced by such activities performed by all involved users together.

During the project evolution stages, GitHub can acquire significant amount of data. This information can present an important characteristics of projects and can be submitted for analysis in order to understand the ways to improve team throughput and consumer satisfaction. However, this data can be difficult to analyze by hand even for the expert in the software engineering field. In addition, it is hard to compare performance of different teams even inside some particular organization, not to mention possibility of such comparison performed between different organizations. This topic is in intensive study today; many researchers try to develop special tools that can help the expert to speed up the analysis and improve quality and accuracy of the feedback [9], [10], [11]. In these works, researchers have used data from issue management system in order to better understand how the team behaves during development stage and what kind of activity can indicate potential problems in the process.

In the work led by Mattila et al., software engineering data from various sources were studied with intention to visualize the development process in order to improve transparency of the actual status of the development and get possibility to react to possible problems faster [9]. This research is based on the two industrial projects of a single multinational Finland-based large-size company involved in software R&D. In this project, researchers used very small data set formed from two cases, thus it lacks generalization. The work described in [10] analyzes a possibility to unify the

engineering data from different system. In the third research by Lehtonen et al. ([11]) the software engineering data from Jira system was used and visualization was carried with intention to study if the sprints hold in the case project [11]. In [9] and [11] works, expert knowledge is required for evaluation, thus limiting application of these methods to large number of projects. Comparison between large number of projects is not possible. For instance, if we would like to compare hundreds of projects between each other. The solution to this problem can be found with the help of machine learning methods, which help to solve various problems without direct human intervension [12], [13], [14]. Using machine learning approach, the necessity for expert participation during estimation stage is kept to a minimum degree, thus, the approach can serve for many users with minimum cost.

1.3 Objectives and main results of the thesis

This work is based on Otto Hylli et al's paper [10], where the team discusses developed Unified Data Model and proposes collection of an issue management data with the help of this model. The collection was possible from wide range of systems, widely used for version control and project management tasks, such as GitHub, GitLab [15] and Bitbucket [16]. Most of the systems offer an Application Programming Interface (API) that can be used during data collection stage. The team analyzed different system used in the issue management work-flow, found and reported the similarities between them, and suggested to unify such data. Finally, they proposed the data and program models for the system. This kind of unification allows one to get the data from different systems in one place and later analyze it using, for instance, one of the pattern recognition methods.

The objectives of this thesis is to research a way to compare quality of projects based on project management data that can be obtained from open repositories. For this work we will collect a large number of projects from Open Source repositories and compare projects applying machine learning methods. We would like to understand differences between these projects in the project management, coding, supporting, communication and other areas. The differences will help to find behaviour patterns inside well and malfunctioning projects.

The main result of the thesis is a developed pattern recognition system for GitHub project data. This system is capable of collecting the data from GitHub into central database in Unified Data Model, finding relevant characteristics, and giving estimates about the project behaviour. Using these estimates it is easier to propose the ways of how teams can improve development process.

1.4 Organization of the thesis

Chapter 2 studies briefly broad concept of the machine learning field. It gives an essential knowledge about the methods that were used in this work. This chapter provides the reasons for choice of machine learning techniques.

Chapter 3 introduces two hypothesis that were tested in the thesis and details implementation of pattern recognition system, which was developed based on the data parsed from 329 project randomly obtained from the GitHub system via it's API. These projects have shared contribution of more than 100.000 users of the GitHub.

Chapter 4 starts from hypothesis assessment. Then it proceeds with analysis that was concluded by evaluation statistical features for projects distributed on the self organized map. At the end of the chapter, we highlighted areas on the map that have distinct characteristics based on overall project performance values obtained during the data collection step.

Chapter 5 evaluates obtained results. We present features of our model that have big influence on project performance and propose future improvements for minor features. At the end we discuss complications that arose in the verification stage.

Chapter 6 presents conclusions and main findings of the research. In addition, we outline the future work to be done, based on obtained results and found complications.

2. BACKGROUND KNOWLEDGE

In this chapter, we present an important background information about technologies and methods that are used in our work. We also present fundamental features and practical advises given by researchers who apply these methods. These remarks allow increasing performance and avoiding common mistakes associated with the application of these algorithms.

2.1 Machine learning

Machine learning is a field of science that study capability of machine to learn. One of the best definition of this subject was given by Tom Mitchel in his book in relation to a computer program:

> A computer program is said to learn from Experience **E** with respect to some class of task **T** and performance measure **P**, if its performance at task in **T** as measured by **P**, improves with experience **E** [17, p2].

Typical categories of machine learning are: regression analysis, classification, cluster analysis and dimension reduction problems.

2.2 Sample and its features

In statistics, a data sample is a set of data that is taken from some population with intention to estimate the characteristics of the whole population. For example, we will consider the binary system, where only "1" and "0" digits exist. The task will be to recognize hand-written symbols that can be either "1" or "0". The samples will be the set of written examples of such digits, produced by different individuals, as shown in Fig. 1. Lets assume that each sample has fixed dimensions, for instance 30×30 pixels.

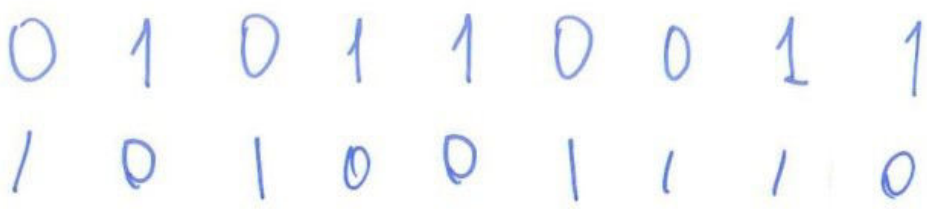

Figure 1: Sample set of written symbols, produced by different individuals

Then each sample can be represented by vector of 900 (e.g. 1×900) different characteristics. These characteristics are called "*features"* or "*attributes"* of the sample.

In our work we will collect raw project data from online repositories. The raw data will be transformed in feature vectors. Where individual features project will reflect different project management areas and quality of team work for corresponding project.

2.3 Learning Paradigms

We will discuss here two major learning paradigms: **supervised** and **unsupervised** learning. The choice depends on a particular learning task and characteristics of available data.

2.3.1 The supervised learning

The **supervised learning** is the task of approximation a function from labeled training data, where the training data is usually a set of training vectors with associated target (desired) output. Algorithm analyzes input and desired output data and produces parameters for approximated target function. The function can be used for estimation of input feature vector to target output.

Usually, some of the instances cannot be directly seen from the target data set. The algorithm should be capable to generalize behaviour in such unseen cases from the training data and produce feasible results. The supervised learning is used for the regression and classification algorithms.

2.3.2 The unsupervised learning

The **unsupervised learning** deals with the problems where a training examples do not have any associated target labels. In such a way, algorithm do not have any measures for success or reward that can help in evaluation of current solution. It is the main difference between supervised and unsupervised learning.

Unsupervised learning algorithm tries to find some hidden structures in the unlabeled input data set. Based on these structures, it distinguishes between potentially different clusters that are present in the given data set.

2.4 Regression analysis problems

Regression analysis is a tool that helps to investigate relations between depended variables and one or more independent variables. For example, in the case of forecasting of sales of the laptops, the relations could be found between the input variables that can

consist of the cost of the laptop, the current sales per week and an age group of buyers, while the output variable is the number of sold laptops for the next week per each age group.

Training data consist from input variables together with associated output variable. Since the input variables and desired output of the sample is known beforehand, this method belongs to *supervised learning* [18]. The estimation target function of the independent variables is called *regression function*.

2.5 Classification problems

The main difference between regression and classification is that in classification problems function assumes to be discrete and has finite number of outcomes, whereas regression assumes continuous function. In classification, we need to choose to which class from a finite set of classes a new observation belongs. The training set includes a data for which a target class memberships are known, therefore this method belongs to *supervised learning*.

Class assignment can be performed by comparing similarity or distance between the sample set and the classes. The class with the smallest distance will be assigned to the considered sample. The similarity measure can be, for instance, a Euclidean or any other distance. The Euclidean distance between points p and q is the length of the line connecting them. For example, if $p=(p_1, p_2, \ldots, p_n)$ and $q=(q_1, q_2, \ldots, q_n)$ are the two point in Euclidean n-space, then the distance from p to q or vice verse can be found using equation 1.

$$d(p,q)=d(q,p)=\sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\ldots+(q_n-p_n)^2}=\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \qquad (1)$$

2.6 Cluster analysis problems

The cluster analysis problems consist of problems where the task is to group input data into a set of clusters. The main principle can be outlined as follows: in a single cluster data exhibit high intra-cluster similarly and, at the same time, low inter-cluster similarly between samples in any different clusters. In other words, the data that was put in one cluster is more alike or has similar characteristics, whereas there is strong difference between the data samples that were put in the different clusters.

Unlike the classification problem, cluster labels and number of clusters are not known beforehand in the training data set. Thus, it is *unsupervised learning* problem. Usually,

researcher has to define an approximate number of classes, which he desires to discover in the given data.

2.7 Dimension reduction problems

Dimension reduction is the process where the number of random variables under consideration is reduced in order to obtain a set of uncorrelated variables [19, p99-100]. This process can be divided into two distinct categories, that can be used separately or together: **feature selection** and **feature extraction**.

The **feature selection** is a process where relevant features are selected for further analysis. The **feature extraction** is a process where from initial features set we obtain reduced set that is nearly as informative and, at the same time, non-redundant as the initial one. Fig. 2 shows schematically the difference between these 2 approaches. The main difference is that for the **feature extraction** whole initial data set is needed, whereas the **feature selection** reduces number of features and can benefit from reduction in the future data collection costs, because less features have to be collected and needed for the analysis (some irrelevant features will be permanently excluded before the training stage).

Main advantages of **feature selection** methods:

- Reduce computational and storage requirement.
- Reduce cost of future data collection.
- Reduce classifier complexity and allow to better understand the data, as the number of features reduced.

Main advantages of **feature extraction** methods:

- Reduce bandwidth of the original data.
- Improve classifier.

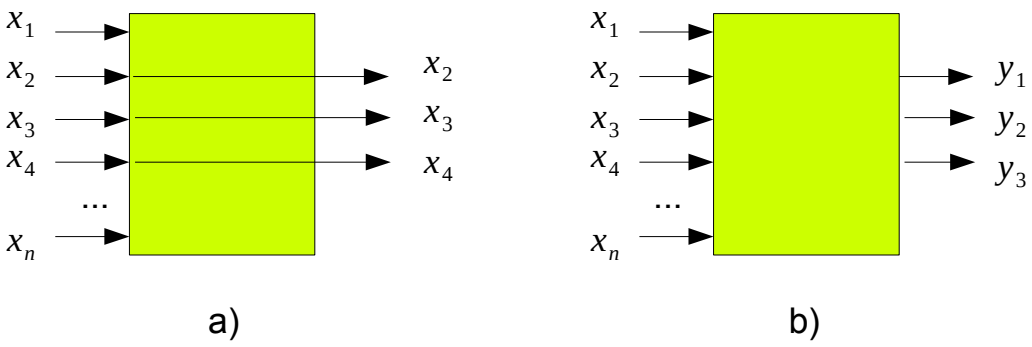

Figure 2: a) feature selection, b) feature extraction

- Give possibility to visualize the data in low dimensionality, for example, in 2D or 3D [20].

These two methods have many benefits and we will use both of them in our work.

2.8 Artificial Neural Networks

In machine learning and pattern recognition field, an Artificial Neural Networks (ANN) represent set of different models that are inspired by biological neural network systems. We will start from easy to understand concept of single neuron and will proceed to complex structures of artificial neural networks.

2.8.1 The single neuron

This relatively simple model was inspired by biological learning system and its ability to learn and approximate information that it receives. The artificial neuron is approximation of real biological neuron. Fig. 3, shows the typical neuron model used in ANN, where:

- $x_0 - x_n$, are input signals.
- $\omega_0 - \omega_n$, are synaptic weights of the neuron, that will be applied to corresponding input.
- φ , is an activation function.
- O , is output.

Where ω_0 is a threshold and x_0 is a constant for this threshold equals to 1.

The output of the single neuron can be computed by the following formula:

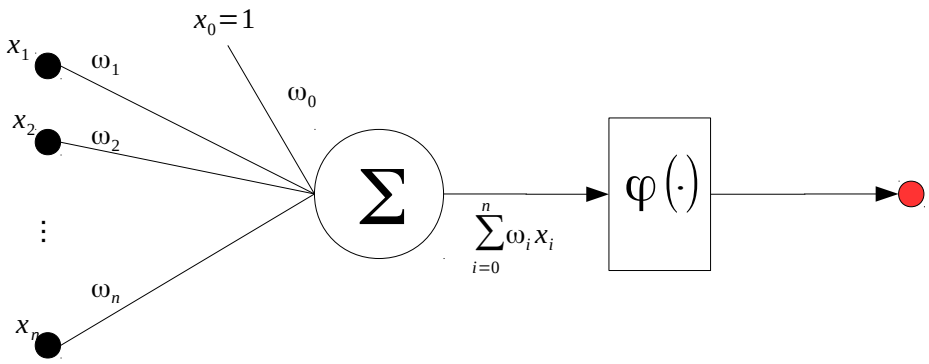

Figure 3: The sigmoid threshold unit

$$O = \varphi\left(\sum_{j=0}^{n}\left(x_j\,\omega_j\right)\right) \tag{2}$$

The input signals represent synapse connection between different neurons or some senor that feed to neuron information for analysis. Weights ω_i, $i = 1, 2, \ldots, n$ are real value constants they determine contribution of particular input to the total neuron output. Values of these constants are determined during a learning stage. Weight ω_0 is a threshold of a neuron, it shifts the decision boundary away from the origin, and does not depend on the input. For example, we look at *binary* classification problem, where only 2 classes exist: *positive* and *negative*. The threshold is applied to a total sum of all its weighted inputs. If the sum is greater than $x_0\omega_0 = 1*\omega_0$ the sample is classified as positive class, otherwise it is negative class.

The activation function, or so called squashing function, maps large value input to smaller output. These functions might be:

- Threshold function (not to be confused with threshold of the neuron, described earlier):

$$\varphi(x)=\begin{cases} 1 & x\geq 0 \\ 0 & x<0 \end{cases}$$

- Sigmoid functions, include functions that have “S” shaped form, for example, one of this type of functions is logistic function:

$$\varphi(x)= \frac{1}{1+e^{-\alpha x}}$$

Plots of these function can be seen in Figure 4, where a) is Threshold function and b) Sigmoid function (logistic function). The functions in the second category have an important features:

- It is increasing monotonically from 0 to 1,
- It is differentiable and continuous in all possible intervals, this is the important property used for ANN trained with back-propagation (BP) algorithm.

The single neuron is able to represent a simple Boolean function, such as: AND, OR, NAND (¬AND) and NOR (¬OR). Unfortunately, it cannot represent such function as XOR, whose value is equal “1” in the case when $x_1 \neq x_2$ and “0” in the case when $x_1 = x_2$. The difference between these functions is in the ability to be linearly separable as shown in Fig. 5, where a) shows linearly separable examples of some function with decision surface represented by the red bold line and b) shows linearly non-separable XOR function output. The decision surface defines the boundary

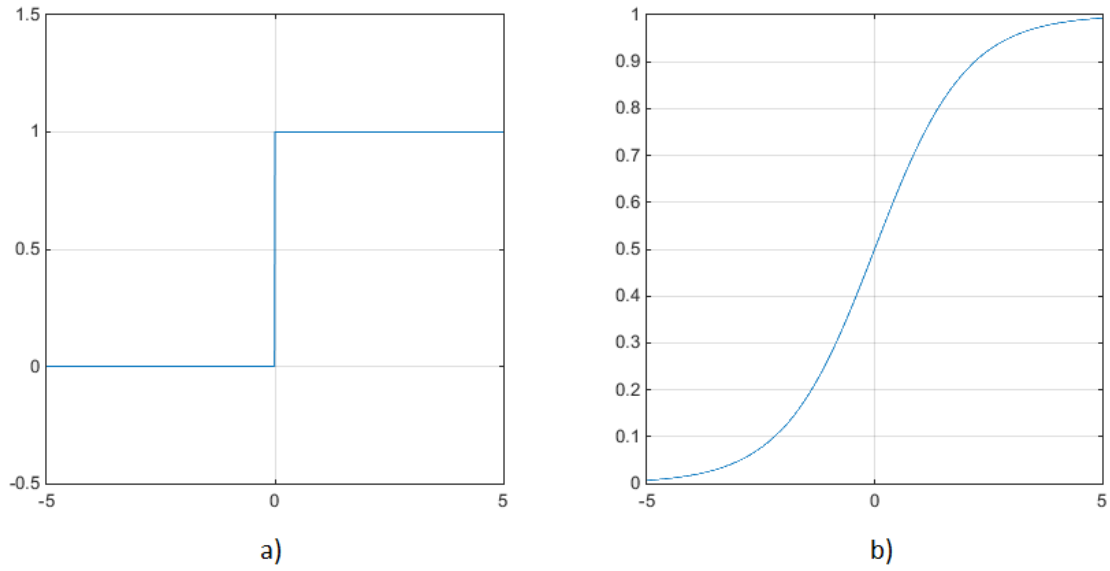

Figure 4: Activation functions

for division between positive and negative samples. The second example cannot be separated by any singular line [17, p87-88]. This kind of restriction can be surpassed with the help of network of neurons. For instance, the solution for XOR problem will include the network with two levels of neurons, where these levels will be inter-connected with each other. It means that the output of first layer of network will be fed to the input of the second one. Generally, hard problems need the multilayer neuron networks in order to obtain feasible solution.

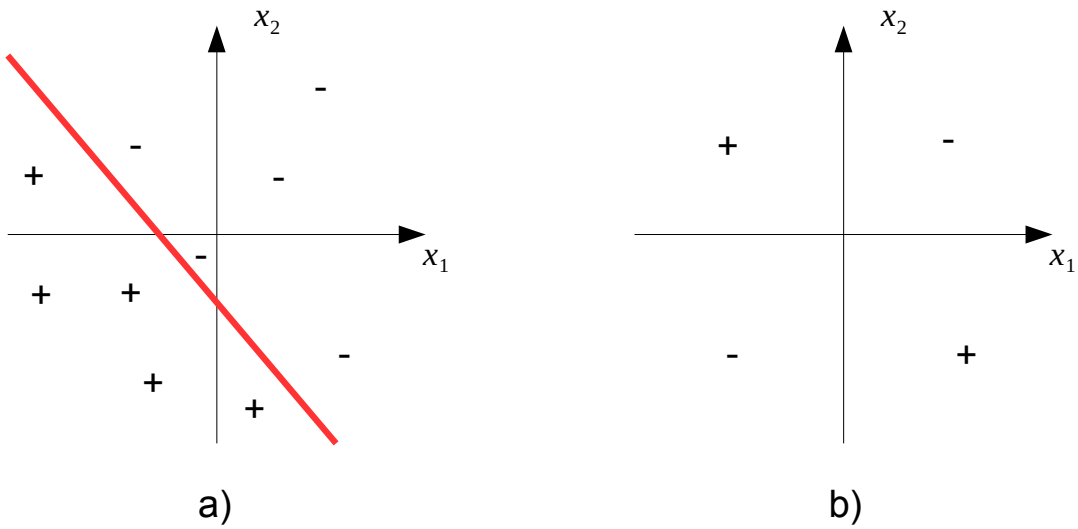

Figure 5: Examples of a) linearly separable and b) linearly non-separable sets. Positive examples are indicated by "+" and negative "-".

2.8.2 Multilayer feed-forward neural networks

We mentioned in the previous section 2.8.1, multilayer neural networks can cope with complex problems. The network can have an arbitrary number of layers with an arbitrary number of neurons in each of them. For example, Fig. 6 shows scheme of typical feed-forward neural network. Feed-forward states for the type of network where input signal propagates forward, from the input towards the output. It has one input layer with three neurons, one hidden layer with five neurons and one output layer with two neurons. In all layers, every neuron is inter-connected to all neurons of the next layer via synaptic connection. The neurons in the first input layers send input data through synapses to the next, hidden layer. Similarly, the hidden layer send information to the output layer. More complex system will tend to have more layers, as they will have possibility to create internal representations and learn different features in each new level [21]. The learning of such artificial neural networks can be performed using error back-propagation algorithm together with some of the optimization techniques, e.g. the *gradient descend*. During the first step it calculates the gradient of a cost function across all weights of neurons in the networks. In the second step, calculated gradients are used in the optimization method that uses it for neurons weights update, in order to minimize the chosen cost function. The cost function is a function that evaluates values of one or more variable of some event and outputs a real number that represents the cost, associated with this event. For example, for a scalar parameter θ ,

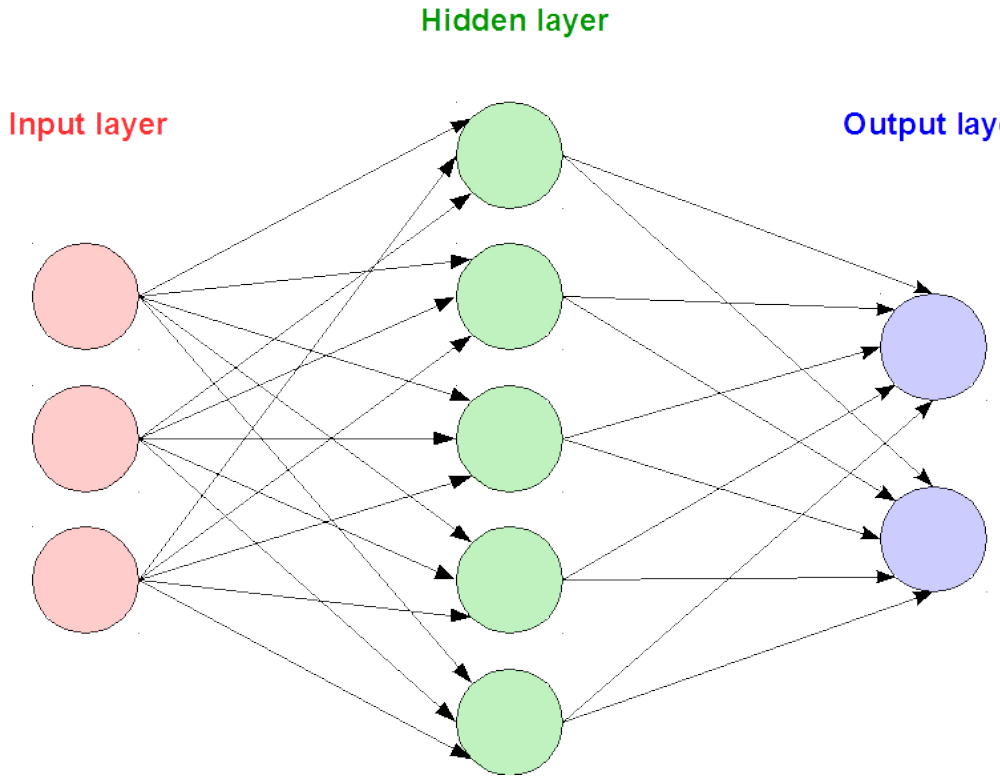

Figure 6: Feed forward artificial neural network with three layers of neurons

and estimates of it $\hat{\theta}$ received with help of some function, quadratic cost function will be:

$$L(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2$$

The back-propagation algorithm can be divided into three steps: initial initialization, propagation and weight updates [22, p447-448]:

- Initialize the synaptic weights and neurons thresholds with a small random numbers.
- Training patterns propagate forward through the network and generate the propagation's output. Backwards propagation of the propagation's output through neural network using the training targets that will output the deltas (local gradients), where deltas are the difference between the input and the output target values of all hidden neurons.
- Multiplication output delta and input activation to get the gradient for each weight. Subtraction of some ratio of the gradient from the weight. The ratio influences the speed and quality of the training and called the learning rate.

The steps two and three will be repeated until overall performance of the network will not meet the requirements. The back-propagation algorithm requires the activation function of all neurons to be differentiable.

2.9 Competitive learning

The **competitive learning** is a form of unsupervised learning in the artificial neural networks. The rules for this kind of learning were formulated by Rumelhart and Zepster in 1985 and later were reformulated by Haykin [22, p58-60]:

- A set of neurons that are all have the same structure with exception to some randomly distributed synaptic weights, and which therefore respond differently to a given set of input patterns.
- A limit imposed on the “strength” of each neuron.
- A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron per group, is active (i.e., “on”) at any given time. The neuron that wins the competition is called winner-takes-all neuron.

These rules will produce individual neurons that will tend to “specialize” on the set of similar pattern characteristics, inherited in the input data. The response of the network

will be the location of winning neuron, point out by one individual “firing” neuron at any time.

2.10 The Self-Organizing Maps

2.10.1 Overview

The Self-Organizing Maps (SOM) is a stand out type of artificial neural networks. For the training of this kind artificial neural network input samples do not need any target values. The SOM is based on competitive learning, where neurons compete against each other for the right to be “fired” or activated, in such a manner that only one neuron of this network is activated at any given time. This particular activated neuron is called “*the winning neuron*” or “*best matching unit*” (BMU). One of these types of network was introduced by Finnish professor Teuvo Kohonen in 1982, sometimes it is called Kohonen map [23]. This model captures salient features that are present in the input data. During training stage, neurons becomes spatially tuned to various input patterns in the course of competitive learning process [22, p453]. Self-organizing map is inherently nonlinear and can be viewed as a non-linear generalization of principal component analysis (PCA) [24].

In SOM, neurons are placed at the nodes of lattice, which is usually one or two dimensional. Higher-dimensional maps are also possible, but not so frequently used [22, p443]. For example, two-dimensional 4×4 neurons Kohonen map is shown in Fig. 7. Every neuron is fully connected to all input features, black lines do not indicate

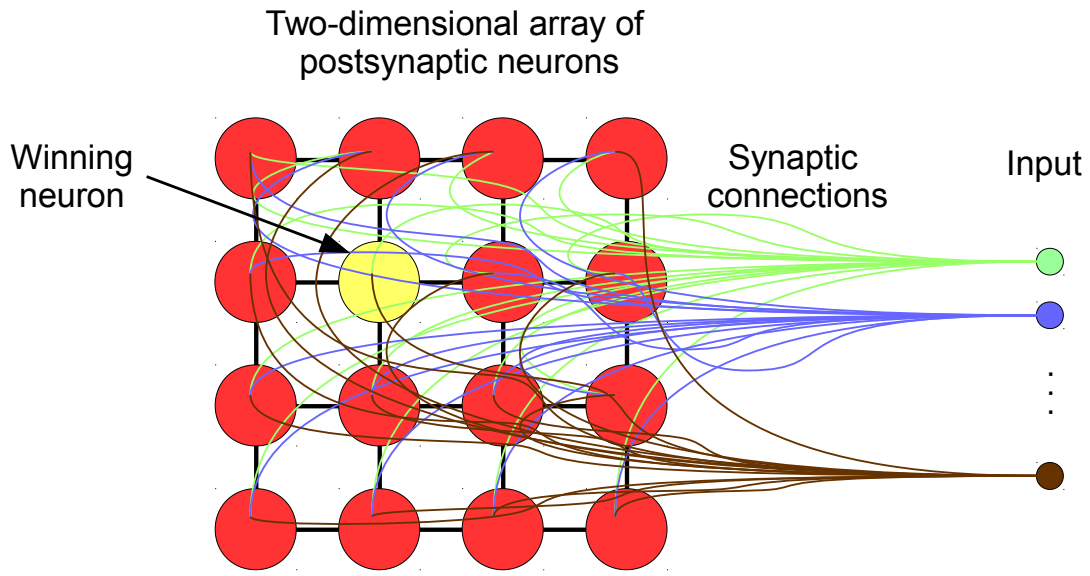

Figure 7: Kohonen map or self-organizing map, as preferred by Teuvo Kohonen [25]

connection between different neurons, instead, they highlight local neighborhood

between them. The "*winning*" or "*fired*" neuron can be distinguished by the yellow colour.

The principal goal of SOM as formulated by Haykin [22, p444]:

> ...is to transform an incoming signal pattern of arbitrary dimension into a one- or two-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion.

During the training stage, algorithm responsible for neuron weights formation proceeds with weights initialization by small random numbers. Then, training stage can be outlined by three fundamental steps, these steps characterize self-organizing map formation **competition**, **cooperation** and **synaptic adaptation** [22, p447-448]:

1. **Competition**: For each of an input samples, neural network computes discriminant function. These functions are the basis for competition between neurons. One particular neuron with largest value of discriminant function called "the *winning neuron*".
2. **Cooperation**: The "*winning neuron*" determines the basis for the spacial neighborhood of excited neurons (whose weights will be affected by following change), this stage provides essential cooperation means between all neighbour neurons.
3. **Synaptic adaptation**: Excited neurons increase synaptic weight values in relation to the input sample in such a way, that next excitation by any similar input sample will be enhanced.

2.10.2 The SOM algorithm

The three basic steps, mentioned in the previous section, can be expanded and complemented in order to better understand how the algorithm works [22]:

1. Initialization: choose small random values for the initial weight vectors $\boldsymbol{w}_j(0)$ of neurons $j = 1, 2, \ldots, l$, where l is the total number of neurons. Weight vectors must have different values for all neurons. The total number weights of neurons corresponds to the number of features, used in the training samples.
2. Sampling: From input space take a sample vector $\boldsymbol{x}$. The vector has m dimensions by the number of different features and is used as activation pattern, applied to all neurons in the lattice.
3. Similarity matching: With help of vector $\boldsymbol{x}$, find the best-matching neuron $i(\boldsymbol{x})$ at time step n, by utilization of the minimum-distance Euclidean criterion:

$$i(\boldsymbol{x}) = arg\ \min_{j} \| \boldsymbol{x}(n) - \boldsymbol{w}_j \|, \quad j = 1, 2, \ldots, l$$

4. Weight update: Adjust the synaptic weight vectors in local neighbourhood, by applying:

$$\boldsymbol{w}_j(n+1) = \boldsymbol{w}_j(n) + \eta(n) h_{j,i(\boldsymbol{x})}(n)(\boldsymbol{x}(n) - \boldsymbol{w}_j(n)) \quad ,$$

where $\eta(n)$ is a learning rate parameter and $h_{j,i}$ is a topological neighbourhood function.

5. Next iteration: repeat steps 2-5, until formation of the feature map has completed.

Now we will consider steps 2-4 with more details in mathematical sense.

2.10.3 Competitive process

Assuming that we have m different features in our samples, our input vector can be denoted by: $\boldsymbol{x} = [\, x_1, x_2, \ldots x_m \,]^T$, synaptic weight vector length of all neurons in the network will have m dimensions. The synaptic weight vector j can be denoted by $\boldsymbol{w_j} = [\, w_{j1}, w_{j2}, \ldots w_{jm} \,]^T$, $j = 1, 2, \ldots, l$, where l is the number of neurons that are used in the network.

In order to find the Best Matching Unit (BMU), we need to compare an inner products $\boldsymbol{w_j^T x}$ for $j = 1, 2, \ldots, l$ neurons of the network and select the largest one. As we need to select the neuron with the largest inner product, we will assume that the same threshold is applied to all neurons without any exceptions. The neuron with largest inner product will determine location, where topological neighbourhood of excited neurons (whose weight will be modified together) is to be centered [22, p448]. In addition, SOM neurons do not use activation functions, a comparison is performed directly based on sample features values with neurons weights. Maximization of the inner product is equivalent to minimization of the Euclidean distance between the vectors $\boldsymbol{x}$ and $\boldsymbol{w_j}$. Using index $i(\boldsymbol{x})$ for this identification one can find it by applying the following condition:

$$i(\boldsymbol{x}) = arg\ \min_{j} \| \boldsymbol{x} - \boldsymbol{w}_j \|, \quad j = 1, 2, \ldots, l \quad , \tag{3}$$

where $arg\ min$ is defined by the following formula:

$$arg\ \min_{j} f(j) := \{\, j \mid \forall\, y : f(y) \geq f(j) \,\} \tag{4}$$

In other words, it is set of points j for which $f(j)$ attains its smallest value. The set can be empty, have one element or multiple elements. The requirement (3) provides essential basis for the competition across all neurons in the network for the right to be fired. The response of this network can be either index of winning neuron or synaptic weight vector that is closest to the input vector in a Euclidean sense, this choice depends on the use-case application of such network [22, p448].

2.10.4 Cooperative Process

As it was said earlier, the winning neuron defines the center of topological neighbourhood ($h_{j,i}$) this step provides cooperation basis for the neurons. In practice, a firing neuron excites closer neurons with stronger change than located farther from it. A typical choice for $h_{j,i}$ is the Gaussian function:

$$h_{j,i(\boldsymbol{x})} = \exp\left(-\frac{d_{j,i}^2}{2\,\sigma^2}\right) \quad , \qquad (5)$$

where $d_{j,i}$ is a lateral distance between the *winning neuron* and current neuron in the consideration, and σ is effective width of topological neighbourhood as showed on Fig. 8. Fig. 9 shows an example of BMU in yellow colour and its radial local neighbourhood during some step of the training.

If we will assume two-dimensional space, the lateral distance will be: $d_{j,i}^2 = \|\boldsymbol{r}_j - \boldsymbol{r}_i\|^2$, where $\boldsymbol{r}_j$ defines the discrete vector of the position of excited neuron and $\boldsymbol{r}_i$ defines the discrete vector of the position of winning neuron. Another feature of SOM algorithm

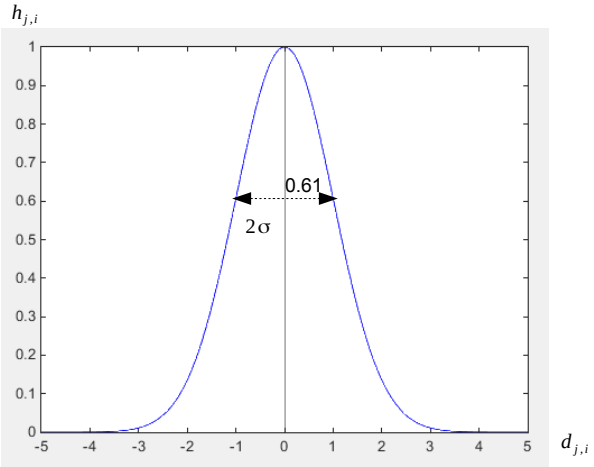

Figure 8: Gaussian neighbourhood function

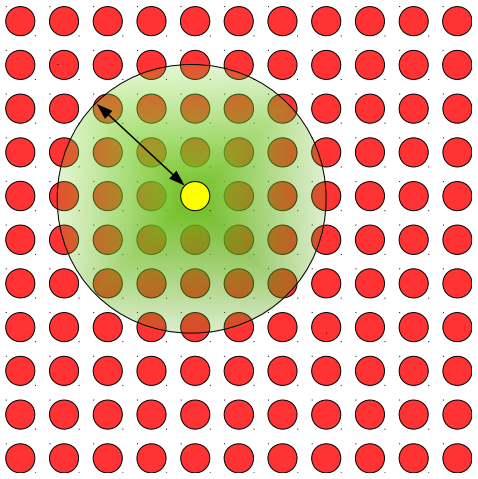

Figure 9: BMU's and its local neighbourhood

is that the size of topological neighbourhood shrinks with time, such dependence can be utilized by the following equation:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right), \quad n = 0, 1, 2, \ldots \ , \tag{6}$$

where σ_0 is the value of σ at the time of initialization of the SOM algorithm, and τ_1 is a time constant, defined during initialization step [22, p449-451].

To sum up equations (5) and (6) topological neighborhood function will be:

$$h_{j,i(\boldsymbol{x})}(n) = \exp\left(-\frac{d_{j,i}^2}{2\,\sigma^2(n)}\right), \quad n = 0, 1, 2, \ldots \ . \tag{7}$$

2.10.5 Adaptive Process

For the network to be self-organizing, the synaptic weight vector $\boldsymbol{w}_j$ of neuron j in the network is required to be changed in relation to the input vector $\boldsymbol{x}$. Given the synaptic weight vector $\boldsymbol{w}_j(n)$ of neuron j at time n , the updated weight vector $\boldsymbol{w}_j(n+1)$ at time $n+1$ is defined by following formula:

$$\boldsymbol{w}_j(n+1) = \boldsymbol{w}_j(n) + \eta(n) h_{j,i(\boldsymbol{x})}(n)(\boldsymbol{x} - \boldsymbol{w}_j(n)) \ , \tag{8}$$

where $\boldsymbol{x}$ is a learning-rate parameter, it must be time-varying and can be defined by formula:

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), \quad n = 1, 2, , \ldots \ , \tag{9}$$

where η_0 is an initial learning rate parameter at the time of initialization and τ_2 is another time constant. This formula is applied to all neurons in the lattice that lie inside the topological neighborhood of winning neuron i . This equation has effect of moving the synaptic weight vector $\boldsymbol{w}_i$ of winning neuron i toward the input vector $\boldsymbol{x}$ [22, p451-452].

2.10.6 Practical advices on how to construct a good SOM

Professor Teuvo Kohonen in his book gave a few advices on obtaining stable, good oriented and unambiguous self-organizing map [25]. The author has highlighted distinct properties of the SOMs and has developed suggestions allowing to avoid some of the

difficulties connected to SOM usage. Some of these suggestions rarely appear in the literature, but can provide powerful effect on the end result.

- Hexagonal grid provides better results with comparison to rectangular in two-dimensional spacing. Although, in the rectangular one is easier to calculate distances and can be thought as native for humans, the hexagonal grid provides better visual inspection, because, it is not favour only horizontal and vertical direction, as the rectangular one. Fig. 10 shows two grids of neurons. The neurons in consideration are highlighted by red colour for each of the grids. The immediate neighbours of these neurons are highlighted by yellow colour. As it can be seen, rectangular grid has only four immediate neighbours, where distance to the neurons laying on the diagonal directions highlighted by green colour will be $\sqrt{2}$.
- Sample reuse techniques and its performance. A complete learning process can take near 100,000 steps. Because the number of available training samples usually is much smaller, the samples must be reused during the training process. There are many alternatives: samples can be drawn cyclically, or in randomly permuted order, or at random from the basic set (bootstrap sampling). Author mentions that in practice approaches do not significantly differ between each other.
- How to enhance important samples that occurs infrequently in training set. As SOM tries to represent probability distribution of applied samples, sometimes it is hard to use this technique in a cases, where important events occur with small statistical frequency. Such infrequent cases must be enhanced during the learning process by taking higher values of learning rate or topological neighbourhood function. Another alternative is to repeat these samples in

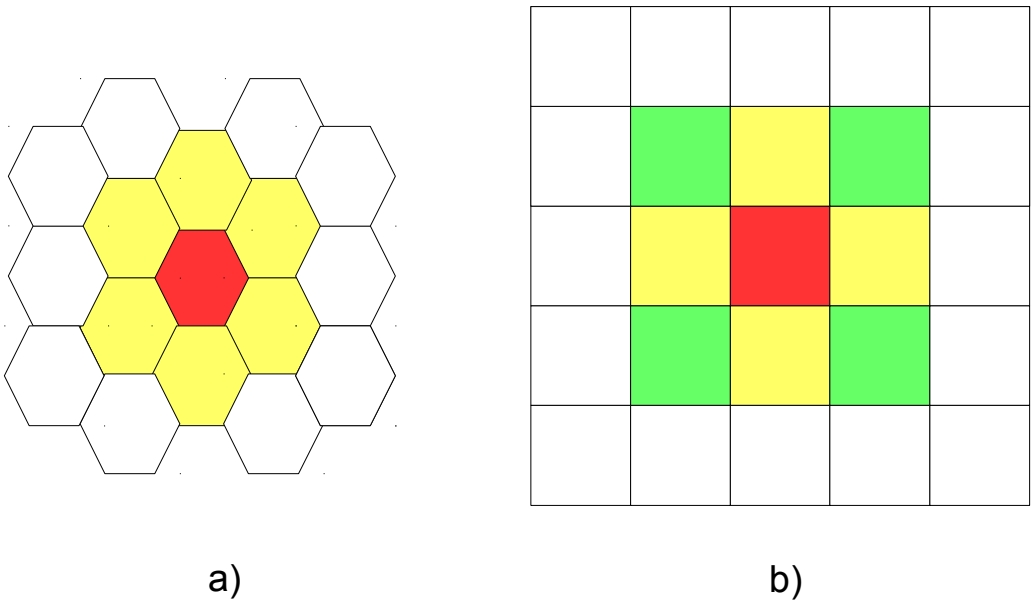

Figure 10: a) hexagonal and b) rectangular grids of neurons

random order a sufficient number of times during the learning process. Determination of proper enhancement in the learning must be done in tight cooperation with end users of these maps, because occurrence of such rare events cannot be easily noticeable.

- Importance of normalization. In practice, feature components of sample set have different units of measurement and can be given in different scales. The performance of SOM will be influenced by such features in samples that have bigger relative values, at the same time, features where samples have smallest values will not provide much influence on the output map. Effective solution in such problem will be the normalization of the variance of each component over the training data. Approach can be especially effective in high-dimensional data. Similarly, one can try to rescale some of the components to provide higher influence for the important feature sets, if they are known, by stretching values of these features.
- Benefits of arrangement usual samples in the center of map. Sometimes it is desirable to map some type of data on a special location, for example, to map some “standard” data into the center of lattice. This approach will force dissimilar samples to be put near the boundaries of the lattice. To do so, one can assign copies of initial values of sample vectors at center locations and keep the learning rate small, for corresponding locations of these vectors, during theirs update operations.
- How to test trained map. The final map depends on several parameters, such as initialization values, the choice of learning parameters, and different sample sequences applied to lattice during training. At the same time, different optimal maps for the same input data exist. The best map is expected to yield the smallest average quantization error, because it is then fitted best to the same data. The latter one, can be determined by $\| x - w_c \|$ via inputting the training data once again after the learning stage, and can be assumed as the important performance index. In such a way, some numbers of random initializations of the $w_i(0)$ and different learning sequences had to be tried before the map with the minimum quantization error will be possible to select.

2.10.7 Properties of the Self-Organized Maps

Some of special properties of SOM are presented here:

- The SOM is capable effectively create spatially organized 'internal representations' of different features of input samples and their abstractions [26].

- The feature map, obtained after SOM training, represented by the set of synaptic vectors in the output space, provides a good approximation to the input vector space [22, p455-459].
- The SOM algorithm is capable to select a set of best features for approximation of given data from input space with non-linear distributions [22, p461].
- The SOM converts the nonlinear statistical relationships between high-dimensional data into pretty simple geometric relationships of their image points on a low-dimensional display. In this way SOM compresses information and at the same time preserves the most important topological and metric relationships of the primary data elements on the display [25].
- SOMs are frequently used as visualization aids [27]. They can help humans easier understand the relation in high-dimensional data. For example, Fig. 11 shows names, SOM hits and boundaries between different classes of species: carnivores, herbivore and birds. Each sample can have a lot of dimensions, and in the case of abundance of the samples, SOM can help to organize them in ordered manner. This organization can help later to find boundaries for different clusters and then to separate different sub-clusters inside a clusters. This can be crucial step, for instance, in the cases where considered data do not have target values or named classes to work with.
- Expression of the dynamic properties of each detail of the algorithm in mathematical theorems is an extremely difficult task [26].

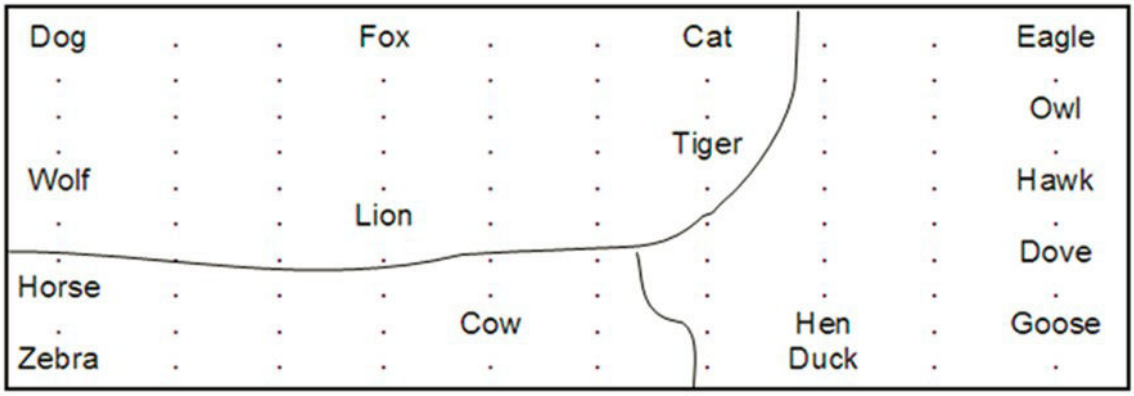

Figure 11: Learned semantic categories in a SOM model [27].

2.11 Summary

The Self-Organizing Map can help to analyze input data and divide it based on some similarities without any prior knowledge and target output. We decided to utilize this characteristics and apply them in our research with intention to understand what are the main differences and characteristics of well and poor behaved projects.

3. METHODOLOGY

This chapter explains detailed steps of our work. First, we look at GitHub work-flow and data characteristics. Next, we present our data model and show data collection process that fills central database system with chosen samples. Finally, the steps used for data transformation, data reduction, and data cleaning will be discussed. These steps are essential in order to prepare collected data for the analysis and to obtain a good learning models during the training stage.

3.1 The Git and GitHub

The Git is a distributed version control and code management system widely used in software development [28]. The GitHub is a web-based Git repository. It incorporates and extends Git functionality. For instance, it has additional feature that allows to bring together different types of users and organize mutual cooperation in a single project work.

3.1.1 Introduction to the GitHub

As was mentioned in section 1.2, GitHub enables communication within developer team and end users of the final product during the development or support stage. The users can participate as testers and report about found bugs, errors or propose extension of functionality, via mechanism of an *issues*. The issues allow users to tell about some important topics that they have in mind and have some relation to the project. In fact, GitHub supports special type of issues called "*Pull requests*", which allow users to modify the source code of the project and propose it to the development team. Later, if developers consider modification to be relevant and important improvement, they will merge the *pull request* into the project. In this way, the mechanism allows to utilize the help from any outer developer. In the GitHub work-flow, any concerned user can make own contribution in the development of interesting product; this improves collaboration links between the developers and the end users and allows to achieve better results together.

In our work, we concentrate on the public available repositories. These repositories have mainly Open-Source projects. However, main principles, ideas and techniques outlined here can be adapted and employed for analysis of the private repositories as well.

3.1.2 Brief outline of GitHub work-flow

In order to better understand the user-developer relations enabled by GitHub, the work-flow procedures should be understood more clearly. This section serves to introduce these procedures and typical activities of users and developers using GitHub communication channels.

The typical work-flow from a **user** perspective can be outlined as follows:

1. Let us assume that a user of some software product finds an aspect that draw his attention. For example, the one that is related to a bug.

2. The user opens an issue with description of all related information. For instance, he describes a set of steps leading to the bug appearance in the product.

3. He can participate in discussion, that clarifies some aspects of an issue or provides additional information.

The typical work-flow from a **developer** perspective can be outlined as follows:

1. Developers find a created issue and analyze it.

2. They label the issue; the labels help to distinguish the application area for the issue.

3. Developers can assign somebody from the team to be responsible for the work on the issue.

4. Team members participate in discussion with the users, who are interested in this functionality.

5. Developers will improve some aspect that is responsible for related issue by a change in source code. If changes are useful, they can be committed to the project repositories.

6. When the issue will be considered to be resolved team manager closes it.

7. If the problem will arise again, then the issue can be reopened and work will be carried on.

3.1.3 GitHub events

From GitHub perspectives, all issues are changed by an events. Any user activity with relation to reference issue is done by means of events. There is certain number of predefined event types used in the GitHub system, which are shown in the table 1.

Table 1: Different event types available in the GitHub [29]

#	Event type	Event description
1	closed	The issue was closed by the actor. When the commit_id is present, it identifies the commit that closed the issue using "closes / fixes #NN" syntax.
2	reopened	The issue was reopened by the actor.
3	subscribed	The actor subscribed to receive notifications for an issue.
4	unsubscribed	The actor unsubscribed from receiving notifications for an issue.
5	merged	The issue was merged by the actor. The `commit_id` attribute is the SHA1 of the HEAD commit that was merged.
6	referenced	The issue was referenced from a commit message. The `commit_id` attribute is the commit SHA1 of where that happened.
7	mentioned	The actor was @mentioned in an issue body.
8	assigned	The issue was assigned to the actor.
9	unassigned	The actor was unassigned from the issue.
10	labeled	A label was added to the issue.
11	unlabeled	A label was removed from the issue.
12	milestoned	The issue was added to a milestone.
13	demilestoned	The issue was removed from a milestone.
14	renamed	The issue title was changed.
15	locked	The issue was locked by the actor.
16	unlocked	The issue was unlocked by the actor.
17	head_ref_deleted	The pull request's branch was deleted.
18	head_ref_restored	The pull request's branch was restored.
19	commit	The commit was made for an issue.

These event types are used during project collection step and they define an important concepts showed in the data transformation part of the work.

3.2 Data collection

In order to perform analysis, we need to collect a data from the GitHub repositories. For that purpose, we decided to look for already existed models, used in software engineering data collection works.

3.2.1 The Unified Data Model

The Unified Data Model was mentioned earlier in the section 1.2, it was proposed in Mattila et al's paper [30]. The model allows to collect software engineering data form various systems. For example, the team implemented parsers for such systems: GitHub, Jira, Bitbucket. In addition, proposed model allows to link related projects together, where the projects can be collected from different systems.

The Unified data model for software engineering data is shown in Fig. 12. The key elements of this model are the **artifact** and the **event**. The description of the model and its implementation in MongoDB, reflects old version of the Unified Model. This version was available during the time, when our research has been started. The current version

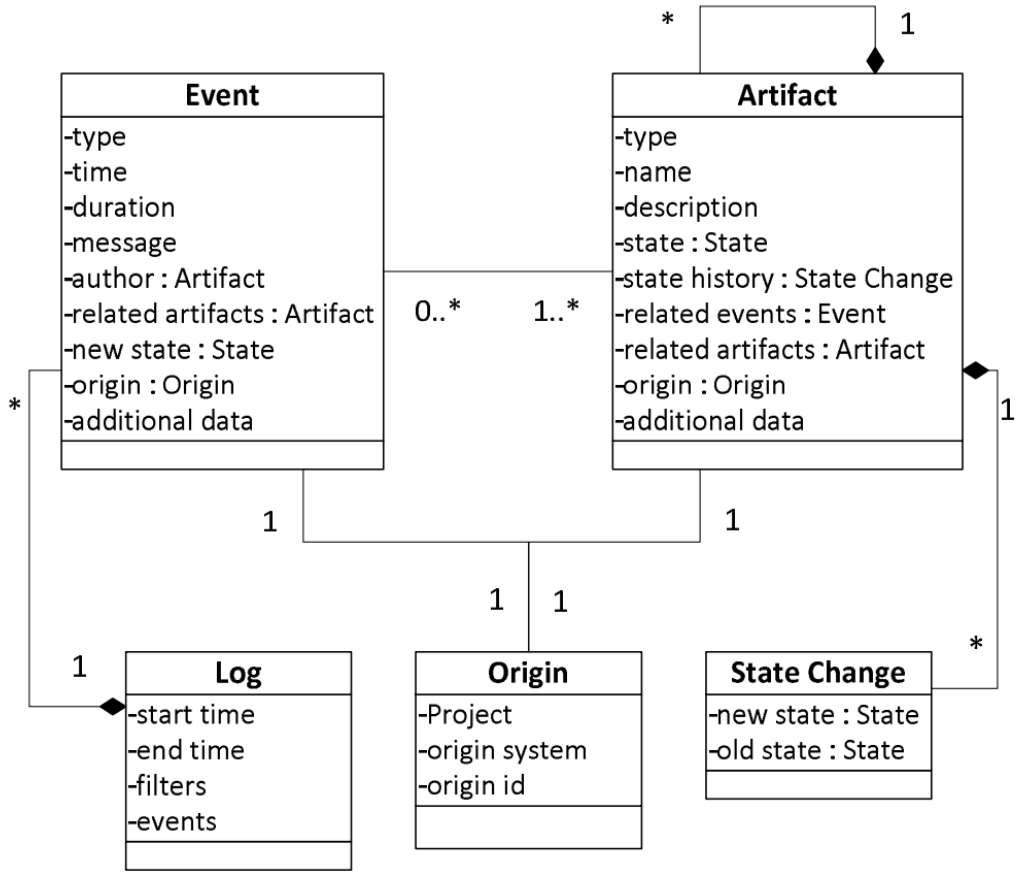

Figure 12: The Unified data model for software engineering data [28].

of the Unified Data Model and its implementation can differ significantly from the one discussed here.

The **artifact** can describe any aspect of software engineering field in the interest of visualization and analysis. They are used to present the objects of the events and can be used, for example, to model an author of the event or they can be a representation of a project developer, source code file, bug or feature ticket.

The **events** are the actions that happen at a certain times and affect artifacts [30]. They can change state of artifact, e.g. to opened or closed. Events have an attributes such as author, time and type of event. Event cannot exist as an entity without the related artifact.

The **artifacts** can be connected to each other and in this way comprise hierarchies, e.g. to form a story, a set of related issues or *test suites*[1]. In addition, this connection can be related to another project or even to a project, that was collected from another system. As with events, the artifacts can have attributes such as type, author, description and related project, described by attribute origin and showed in the Fig. 12 [30]. The origin attribute is used for project identification for all related artifacts. The additional data attribute is used in the both discussed instances (artifacts and events) for storage of additional information that can be collected from different systems and would be out of scope of the Unified Data Model. This is done, in the object format that is capable to store set of predefined data types. Otherwise this information will be discarded. The main purpose of this model is to store whole information present in the system. Later, the data from additional data attribute can be incorporated in the analysis using supplementary examination.

Finally, the data model also has a concept of **log**, which can be used for storage of events that took a place in a certain time frame. The **logs** can be used in data querying to filter the time frame and other attributes, such as origin, type or related artifacts of the events under consideration [30]. The main purpose is to speed-up query search inside database. **Logs** can be created dynamically by user's inquiry.

The architecture of implemented framework is shown in Fig. 13. The framework consists of parsers/linkers, MongoDB database, visualization plugin and REST API. The parsers are used for data collection from different systems. The linker is responsible for bond creation in received events and\or artifact together. It is used for creation of artifact-artifact or artifact-event links. The linking stage is an essential stage for integration of projects obtained from different systems. For example, GitHub was used as version control system and Jira for project management tasks. After parsing and

1 *Test suite* is a collection of test conditions that are intended to test a software product and show that it has specified set of behaviours

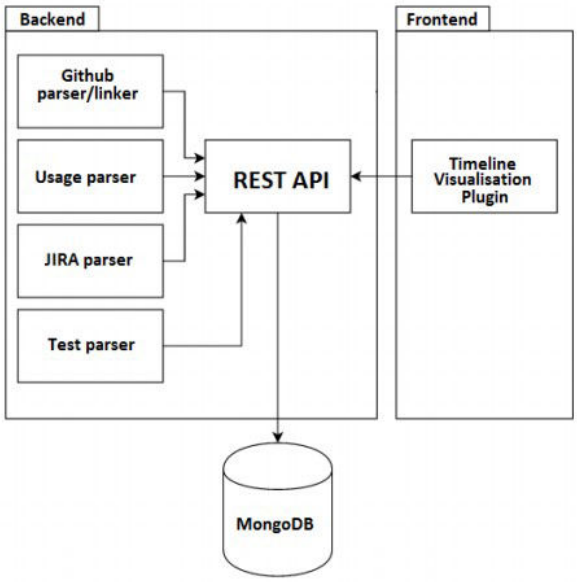

Figure 13: The architecture of program model for data collection, conversion, linking, sorting and visualization

linking stage, data is saved to the database in the Unified format. As we will not use visualization plugin nor server (REST API) in this work, we will not discuss them here. For further information the reader is refereed to [10].

3.2.2 Analysis of the Unified Model and its implementation

For the purposes of our research, the GitHub parser/linker tool and a MongoDB script with database structure description were provided by Mattila's team. At the beginning of this project, the team did not have any information about the current implementation state of the Unified Data Model in MongoDB. In addition, the model presented in Fig. 12 was changed after publication of the work, and new model was not documented. We decided to analyze the model with intention to understand the data and all relevant information that can be extracted from the Unified Data Model. The results of this analysis are shown in Fig. 14. We see that the Unified Data Model, discussed in [30], [10], [31], was changed.

The following notations were assumed:

- +: denotes primary key.
- *: denotes indexed fields.
- -: denotes non-indexed fields.
- # denotes foreign key.

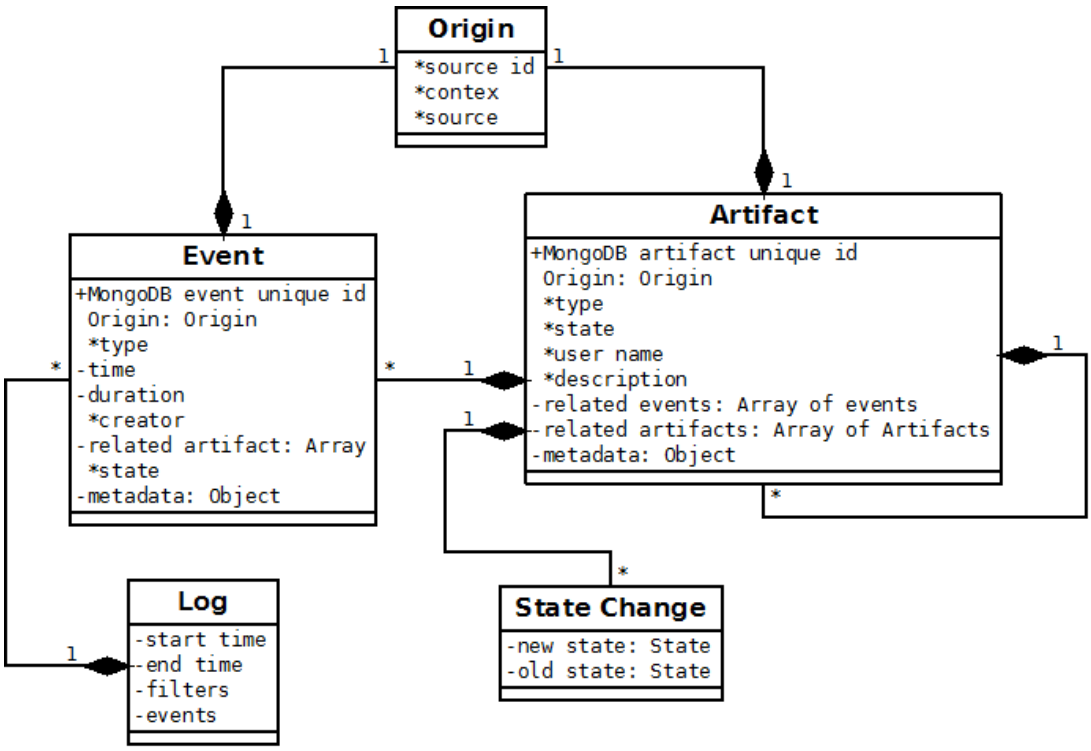

Figure 14: The implementation of database structure, acquired from script description for database creation and analysis of source codes of parser\linker program

The analysis of the implementation uncovered the following issues that can potentially complicate data analysis, increase search time for queries inside database or make the collection of numerous projects difficult.

1. The Origin object shown in Fig. 14 was used as embedded data type for both: artifacts and events instances. For every object of both types a separate instance of Origin will be created. This will decrease the number of queries that are needed to perform during data extraction or update operations for an individual record. The technique multiplies identical data, because every instance of artifact or event will have own copy of Origin. This fact complicates the update operation of such data. In order to change the whole instance, the database has to change many records, related to this instance simultaneously. Such structures drastically increase space, which is required for storage [32].
2. The model did not keep all information that GitHub provides for user. The user related information is currently represented by single string attribute. When the number of collected projects will be large, it will be highly probable that physically different users will have the same name. In this case, they will be indistinguishable from one another in the database. In addition, some of relevant

information given by GitHub, such as email, system user id, and full name are not currently collected. The information related to user details can be highly desirable, for example, during the verification stage. It can give the possibility to ask user opinions about the project quality and quality of team-member work via questionnaires or direct letters.

3. The presence of unnecessary objects. Log and State Change seems unnecessary since all the information that stored in the DB allows to get this kind of information when it is needed.

4. The high level of unification. In some of the cases unnecessary unification can bring to complication during the analysis stage, when many different instances will be kept in one representation. It is hard to keep in mind enormous amount of information about different relations used in the model and needed for analysis. For instance, lets take a look at the concept of user. It is the key object for many systems, for example the GitHub, where every event or issue has creator. It is better to separate often used concepts from the polymorphic artifact as it will help to describe the model easier and with higher efficiency for following interpretations.

5. With high level of unification, many unsupported attributes will end up in meta-data field shown in Fig. 14. In this case, indexing of such attributes will be impossible, which will increase the time needed for access to this information, leading to large time increase required for the analysis.

6. Redundancy in the attributes. We found that some of event and artifact object attributes can be omitted from the implemented model. For example, event has attributes "*related artifact*" and artifact has attributes "*related event*". These attributes create crossed links between each other. For example, in the case where one will find an event (and event has attribute "related artifacts") you can already have access to the related artifact, on another hand with artifact id you can find all related events.

7. The length and type of used indices. The performance of indexed fields depends on the index length, and it is advisable to keep the length shorter. All indices used in original model implementation were text indices with an arbitrary length [33].

8. Relations between different instances that are used in the GitHub work-flow represent relational connections. For example, user-artifact-event connections. They are easier to model with the help of Relational Database Management System (RDMS). The MongoDB and its flexibility allows an easier data collection; at the same time, it can lead to complicated and time consuming data transformations. The transformations must be performed on the raw software

engineering data, in order to obtain a model that can highlight behaviour patterns in the project team. The difficulties that can be found with meta-data field in MongoDB can be easily overcomed with the help of Binary Large OBject (BLOB) fields in RDMS-type databases. These fields can have arbitrary length and structure and can contain, for example, JSON files with any desirable number of data types and length.

9. In some of the cases, original GitHub parser can coincidentally collect an issue and all corresponding events several times. This occasional behaviour multiplies data and can add some bias in statistical results. Relational databases allow to eliminate such situations without any additional effort, naturally keeping uniquness of all records in the database.

3.2.3 The modified model

The points outlined in the section 3.2.2 suggest directions for further improvement of the Unified Data Model and gave hints to implementation of it with help of Relational Database Management System (RDBS). New model included new separate instances, such as user, commit, project artifact label. At the same time, events and artifacts were incorporated with slightly changed structure due to increased number of possible instances in the Modified Unified Data Model. The new model has similar powerful features as the original one that was discussed in 3.2.1. It has a normalized structure that decreases amount of space required for storage, improves index performance, and, as a consequence, search time inside the database. The new model is shown in Fig. 15. The notations have the same meaning as described in section 3.2.2.

The **User** is an initiator of events and artifacts. The **Event** has related instances: "*Event type*" and "*Commits*". "*Event type*" defines acceptable event types across all systems. "*Commits*" can have "*Commit parents*" instances, the ordered sequence reflecting the order that previous commits were applied. The artifacts can reference another artifact, not necessary from the same system as in the original model. They can have arbitrary number of labels that are kept separately. In addition, there is a counter attribute that is capable to count the number of uses of the same label across all projects collected in the database. This feature will help to find most frequently used labels and pay attention to user behaviour in such cases. The **Artifacts** have connection to the **Project** instance that contains project details. The **Project** in its turn has connection to the **Origin** instance that shows the system details, from which this project was collected.

To implement such model, we chose Firebird RDMS [34]. The detailed information about different fields and corresponding description can be found from conceptual

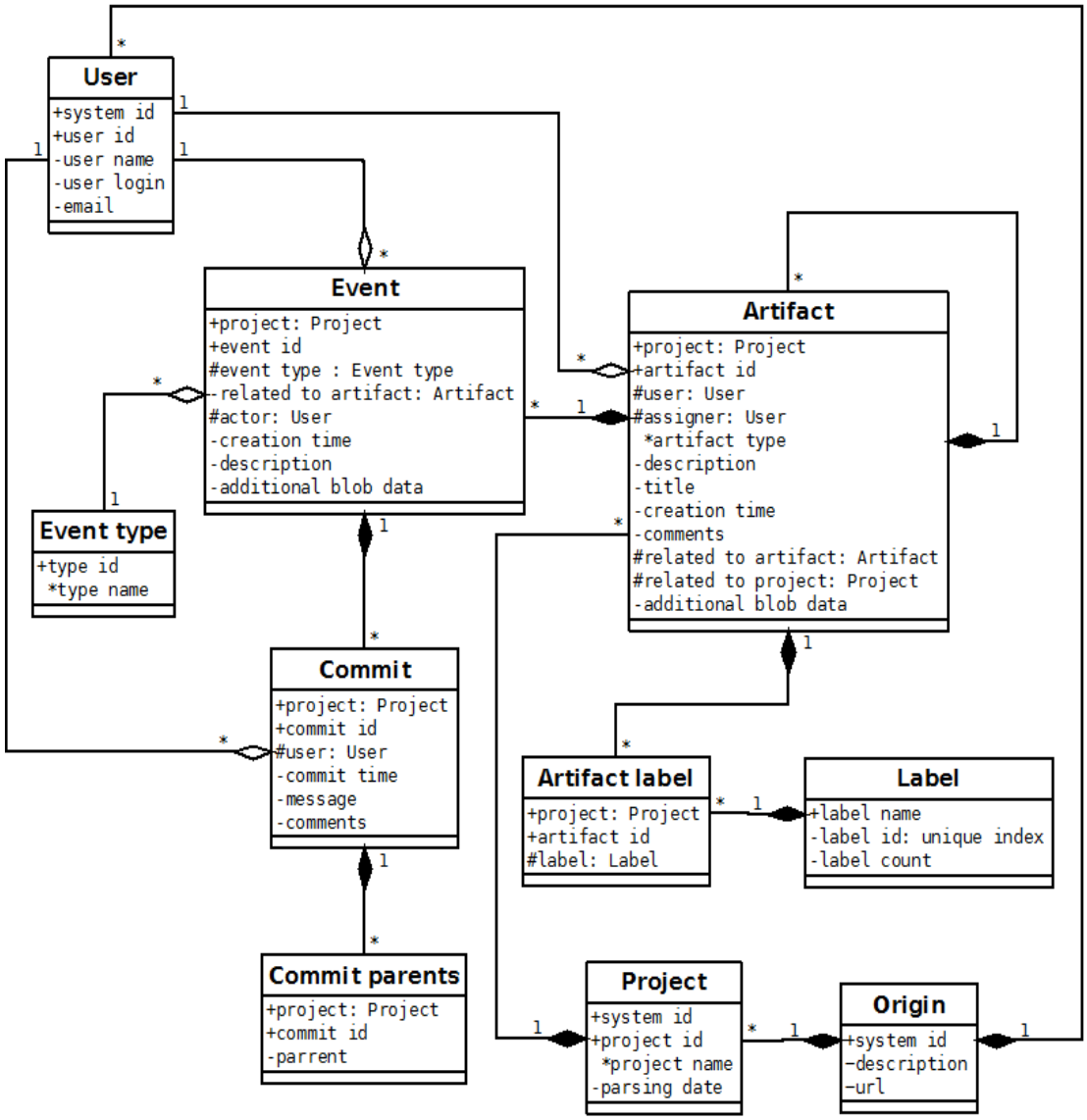

Figure 15: The Modified Model

schema[2] tables 2-11. The tables give information about field types used in the implementation and show different attributes of these fields such as "*Primary key*", "*Foreign key*", "*Unique key*" and *Indexed* attribute, where

- Index is a database structure that improves speed search in the allocated table. It maintains special structure that requires additional operations for maintenance during insertion, deletion and update operations inside database. In exchange, it delivers significantly improved speed for any data retrieval operations. Example of indexed attribute is the "*artifact type*" in instance **Artifact** shown in Fig. 15.

2 A conceptual schema is a high-level description of a business's informational needs. [1]

- Primary key includes one or more attributes in a table. The basic requirement for a primary key is to form value uniquely identifying row in the table. In addition, this attribute cannot have NULL value. For every attribute associated with primary key database server will create an index that allows fast search inside the table. Example of primary key can be attribute "*System id"* of instance **Origin** shown in Fig. 15.
- Foreign key is a one field or a collection of fields in one table that uniquely identifies an attribute from another table. In other words, the foreign key defined in the one table refers to the primary key of another table. In that case the latter table is the parent table for the foreign key. As with primary key, server will create index for foreign key. Example of foreign key can be the attribute "*User"* in the instance **Artifact** that refers to parent table **User** shown in Fig. 15.
- The values of Unique key are required to be unique for each record of table. Example of the unique key is the "*label id"* attribute in the instance **Label** shown in Fig. 15.

For all of the instances, such as user, commit and issue, original GitHub primary keys were used. These keys eliminate the problem with repeated issues, mentioned in the section 3.2.2 guaranteeing uniquness of the record.

Table 2: Table ORIGIN keeps detailed information about different systems

Name	Data type	Description
SYSTEMID	Number	Artificial unique number, used for identification of system for parsed project (Primary key)
DESCRIPT	String	Description of the system
URL	String	Contains URL of the system

Table 3: Table PROJECTS keeps information about project details

Name	Data type	Description
PRJID	Number	Artificial unique number of project, given by database (Primary key)
SYSTEMID	Number	Describe system from which this project was retrieved (Foreign key)
PROJECTNM	String	Name of the project (Indexed)
PRSDATE	Timestamp	Date when project was saved from outer system used

		for in order to determine the length of unresolved artifacts in order to not penalize projects that were started earlier.

Table 4: Table USERS keeps user related information

Name	Data type	Description
USERID	Number	User identification from outer system (Part of primary key)
SYSTEMID	Number	Described system from which user was retrieved (Part of primary key) (Foreign key)
USERNAME	String	User name
USERLOGIN	String	User login name
EMAIL	String	User email address

Table 5: Table LABELS keeps information about label names and number of times each label was used

Name	Data type	Description
LBNM	String	Label description (Primary key)
LBID	Number	Artificial identification given by database (Unique index)
LBCOUNT	Number	How many times this label were found in artifacts

Table 6: Table TYPES describes different types of events

Name	Data type	Description
TYPID	Number	Artificial unique index for event type (Primary key)
TYPENM	String	Type description used for the label

Table 7: Table ARTIFACTS keeps information related to the artifact and corresponding project and links together related artifacts

Name	Data type	Description
PRJID	Number	Project identification (Part of the primary key) (Foreign key)
ARTIFACTID	Number	Artifact identification (Part of the primary key) (Unique)
USERID	Number	User name identification references User table (Foreign key)
ASSIGNID	Number	Identification of assigned user references User table (Foreign key)
ARTTYPE	String	Artifact type: '0'-pull requests, '1'-issue for GitHub related projects (Indexed)
DESCRIPT	String	Artifact description
TITLE	String	Artifact title
ISSTIME	Timestamp	Time the Artifact was created
COMMENTS	Number	Number of comments for this Artifact
RELART	Number	Related artifact the parent for current artifact
RELPROJECT	Number	Related project the related artifact belongs
ADDDATA	BLOB	Additional data in BLOB field type

Table 8: Table ISSLABELS keeps information about labels that were used in artifacts

Name	Data type	Description
ID	Number	Artificial unique number (Primary key)
PRJID	Number	Project identification number (Foreign key)
ARTIFACTID	Number	Artifact identification number (Foreign key)
LBID	Number	Label identification number (Foreign key)

Table 9: Table EVENTS keeps events and connects related artifacts.

Name	Data type	Description
PRJID	Number	Project identification number (Part of the primary key) (Foreign key)
EVID	Number	Event identification number (Part of the primary key) (Unique)
TYPID	Number	Event type identification (Foreign key)
ACTORID	Number	User name identification number references User table (Foreign key)
REFARTFACT	Number	Reference to the artifact that this event is related
EVTIME	Timestamp	The time the event had happened
DESCRIPT	String	Event description
ADDDATA	BLOB	Additional data in BLOB field type

Table 10: Table COMMITS keeps commits

Name	Data type	Description
PRJID	Number	Project identification number (Part of the primary key) (Foreign key)
COMMID	String	Commit identification number (Part of the primary key) (Unique)
USERID	Number	User name identification number (Foreign key)
COMMTIME	Timestamp	Time the event happened
MSG	String	Commit description message
COMMENTS	Number	Number of comments

Table 11: Table COMMPARNT keeps parents for current commit

Name	Data type	Description
ID	Number	Artificial unique number for parent commit (Primary key)
PRJID	Number	Project identification number (Foreign key)
COMMID	Number	Commit identification number (Foreign key)
PARENTS	String	Commit parents

Based on tables 2-11, we wrote SQL scripts for Firebird DBMS. These scripts create database structure and all relating tables, indexes, keys and constrains. The working script is presented in the appendix A.

3.3 The data collection step

We collected 329 projects from the GitHub repositories. An individual project has wide range of created issues, related events and participated users. The number of issues in the projects varied from 15 to more than 10000. Altogether, collected project data was created by more than 10,000 users who participated in the development all round the world. The list of collected projects can be found in the appendix B.

For data collection we modified parser script that was provided by Mattila's team (section 3.2.2). The parser is capable to store received information in the Firebird DBMS with accordance to the Modified Model (section 3.2.3). The framework for collection, transformation and analysis of the data is shown in the Fig. 16. Currently, the implementation consists of the backend part: GitHub parser, Collector program, database implementation and data transformation scripts. Collector program is capable to load list of projects from file and start wide scale collection into database. Database has a set of scripts for data transformation and reduction. The scripts extract all relevant information and submit this information for evaluation implemented in the feedback program. The feedback program extracts statistical information and cleans it from missing values. In the last step, the cleaned data is analyzed by self-organizing map and forms estimated performance of the project with comparison to all other projects that were collected before.

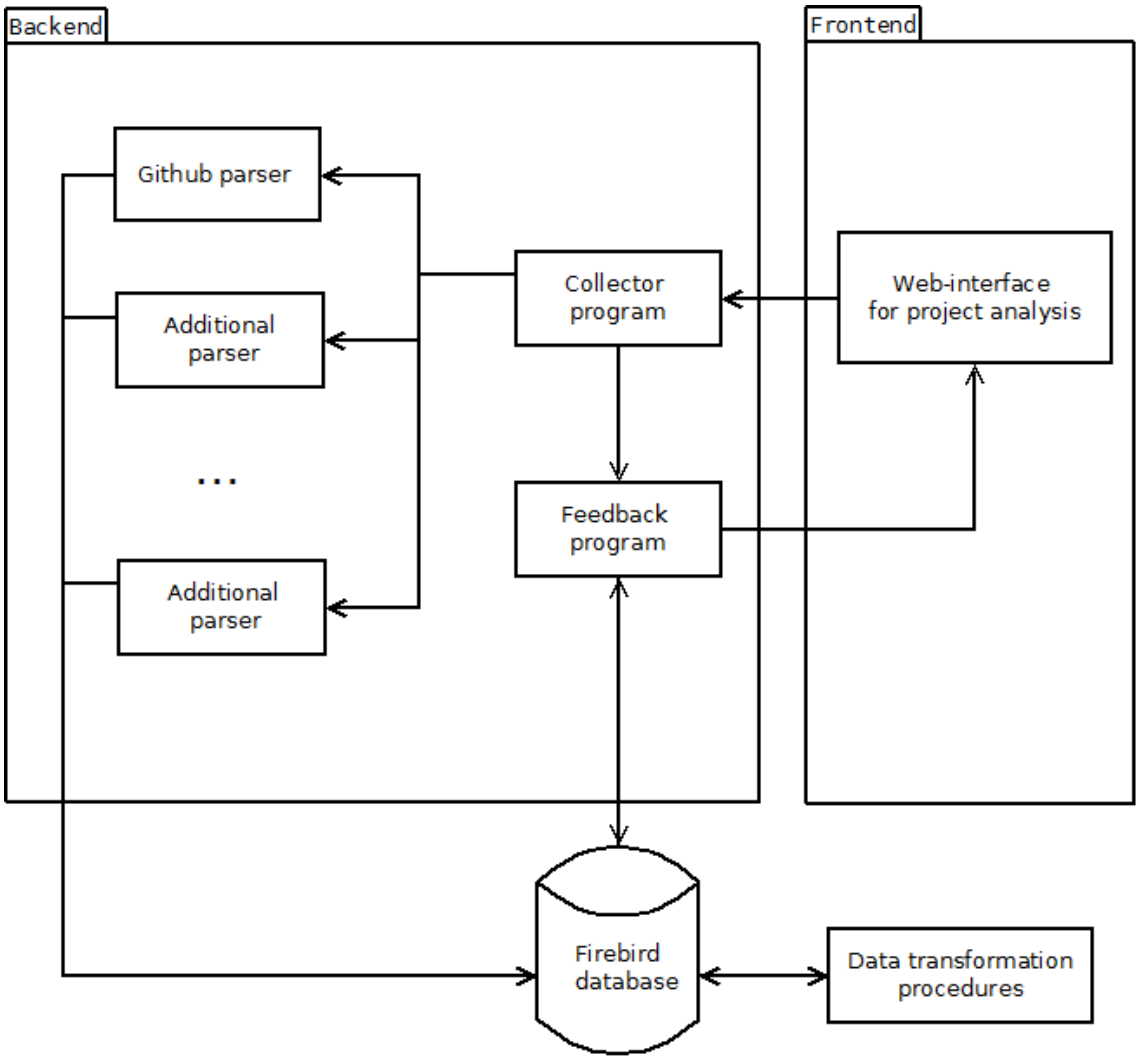

Figure 16: The architecture model

3.4 Summary of obtained software engineering data from the GitHub

As we mentioned in previous section (3.3), our database contains 329 projects. The data from these projects was analyzed with intention to find common properties. During the project collection step we implemented special feature that counts number of uses for each label-issue connection that is present in the projects. Information concerning labels with indistinguishable text is kept in the same record that have a counter. The instance "*LBCOUNT*" from the table 5 stores this number. Based on most frequently used labels we were capable to divide all issues on 4 distinct categories such as: **Bugs**, **Features**, **Documents** and **Others**.

The category **Bugs** combines issues that have relations to errors, flaws, failure or fault reports in a software product. These kind of issues are created in response to some problem that user encountered during work with software product.

The category **Features** combines the issues that have relations to introduction of new features into software or requests from users for extension of functionality or some enhancements. These issues mainly responsible for software capabilities and applicability by the user.

The category **Documents** combines the issues that have relations to writing documentation for software product. In addition, this category includes issues with questions and requests for support.

All other issues that cannot be placed in any of the three previously mentioned categories will end up in the **Other** category. The first three issue categories are very important for most projects and were extensively used across many of them. We decided to treat different categories separately and study team behaviour with relation to actions and length of time, which was used to close issues.

3.5 The data transformation and preparation

3.5.1 The four issue categories

To divide all issues between four previously mentioned categories we wrote a simple stored procedure inside our database. The TMPLABELS procedure divides all issue labels that are related to **Bugs**, **Features**, **Document** or **Other** categories. In addition, this procedure finds labels that are responsible for *won'tfix* and *duplicated* issue labels. The program listing is shown in appendix C. The procedure fills additional auxiliary table in the database. Conceptual schema table for the procedure is shown in the table 12.

The simplicity of the stored procedure is an important feature enabling simpler debugging process. Debugging will be needed, if labels of newly collected projects will be put in wrong categories. This procedure need to be executed before new estimation will be possible. As it can be seen in from the listing of the procedure, the main separation is performed with the help of just few lines of code for each of the categories and can be easily rewritten later.

Table 12: TMPTLABELS keeps category markers for labels used in issues

Name	Data type	Description
TLGID	Number	Artificial unique number (Primary key)
TLBNM	String	Label identification number used for determine the label type (Primary key)
TLBID	Number	Reference for LABELS table (Foreign key)
REALNAME	String	Computed by field shows real label name from LABELS table

The output division performed by the procedure is shown in the table 13. The table presents all different labels parsed for all three categories. In the header of the table are presented category names. Each column shows all label descriptions found for categories. In the case when there is no any category for a label of an issue, the issue will be put in the **Other** category. As it could be seen from table 13, the procedure was capable to choose and mark related labels correctly across all 329 projects. This procedure has to be executed each time a new project was added in the database.

Table 13: Label divisions by categories performed by stored procedure in database

#	Bug	Feature	Document
1	bug	enhancement	documentation
2	bug-vim	feature request	question
3	security	Feature request	doc
4	confirmed-bug	Dev Feature request	help wanted
5	Bug	NEW FEATURE	support
6	Bug Report	Feature Request	NEEDS HELP
7	browser bug	Feature	Question/Help
8	bugfix	feature	Documentation
9	t2:defect	Enhancement	help-wanted
10	Confirmed Bug	New Feature	deprecation-help
11	Apple Bug	t1:enhancement	docs
12	bug?	features	FAQ
13	confirmed bug	task	Help needed
14	Bug report	Community Task	Question
15	[t] Bug	feature-request	Help appreciated
16	Defect	[t] Feature	Add support
17	Bug (confirmed)	Task	helpmeplz
18	Bug (to verify)	category:enhancement	faq-able!
19	category:bug	category:feature	documentation issue
20	downstream problem?	type:enhancement	needs documentation on wiki
21	type:bug	development tasks	Doc
22	bugfixing	easy task	docs-and-samples
23	memory problems	I-enhancement	category:question

#	Bug	Feature	Document
24	metabug	Kind:Enhancement	type:docs
25	A-security	Kind:Feature	status:help wanted
26	Kind:Bug	Type-Enhancement	help
27	Note:Erlang bug	Type-Task	Documentation and Guide
28	Security	Analyzer-NewTaskModel	A-docs
29	bug (divergence)	Features	Kind:Documentation
30	bug (build)	feature enhacement	HelpWanted
31	Type-Defect	Request: new feature	docsystem
32	bug (browser)	Request: enhancement	Area-Documentation
33	bug (pixi)	PR: New feature	Docs-LanguageSpec
34	bug (phaser)	new-feature	Type-Documentation
35	bug (3rd party)	enhancements	Docs-UpAndRunning
36	Browser Bug	task-todo	Docs-StyleGuide
37	BUG	task-done	Docs-API
38	good first bug		Docs-Tools
39	accepted bug		Docs-CodeLab
40	plugin problem/fix		Docs-Tutorials
41	possible angular bug		X Deprecated Docs-Editor
42	bugsnag		Docs-Polymer
43	Minor Bug		Docs-Articles
44	Major Bug		Docs-Synonyms
45	Security Issue		Docs-Pub
46	jQuery Bug		Docs-Cookbook
47			Docs-FAQ
48			Pkg-CollectionHelpers
49			Docs-Spec
50			Docs-Requested
51			question for user
52			Docs
53			Support
54			Docs Issue
55			help needed
56			PR: Docs
57			Documentation etc
58			User support
59			Website and Docs
60			Help wanted
61			Help (please use Stackoverflow)

3.5.2 The analysis of issue work-flow

As it was mentioned in section 3.1.2, an issues follow certain work-flow during their lifetime. Any issue starts from formalization problem, following by possible labeling and user assignation, discussion, improvement steps and etc. The work-flow hopefully ends with issue closing event. These steps form a complete life-cycle of an issue. The length of the time an issues of project keeps open and the ratio of closed issues to total number of issues are the important factors of project performance. These factors can show how the team was devoted to the development of the project and can potentially influence user satisfaction.

In order to extract the presence of events that can help to highlight problem in project management we wrote additional procedure. This procedure extracts existence and length of certain facts from issue history. The **EVENTPARSER** procedure is responsible for this extraction and outputs the data for each issue of the projects under consideration. The output parameters are shown in table 14. The source code for this procedure is shown in the appendix D.

Table 14: Output parameters for EVENTPARSER procedure

#	Name	Description
1	ISSID	Issue identification number
2	ISTP	Issue type
3	LGTH	The length between the issue was opened and closed on the time of data collection
4	COMMITNUMBER	The number of commits
5	RPTM	How many times the issue were reopened
6	RPLGTH	The length in days the issue was reopened
7	N_COMM	The number of comments
8	ASSGN	The identification number of responsible user
9	DUPL	Was this issue duplicated
10	PPART	How many people participated in the work on the issue
11	EVENTSNUM	The number of events created for the issue
12	LABELED	Was the issue labeled at least with one label
13	FIRSTLABEL	The time between the issue was opened and first label was assign for it
14	CLOSED	Was the issue resolved at the date this project was collected

The GitHub has special issue type "*Pull request*". It is created by users who decide to improve project and write own commits. The *pull requests* can be merged in the project and incorporate changes. We decided to study how frequently this type of issues were merged in the projects. For that purpose, another stored procedure the **PULLREQUESTS** was written; it analyzes pull requests that are present in project and outputs "1" if issue merged or "0" if not for every "*pull request"* issue of the project.

Finally, we analyzed how many commits per active user were made. The last procedure **COMMITSPERUSER** analyzes each active user of the project and outputs corresponding number. By the active user we assume a user who made any contribution during project lifetime. The source code for this procedure is shown in the appendix E.

3.6 The feature extraction

3.6.1 The statistical quantities used in feature extraction

An individual project can have an indefinite number of issues, issue labels and people, who participate in the work. To unify all projects characteristics we derived an initial pool of features that consists from simple statistical quantities. The initial pool is shown in the table 15. We did not consider many different quantities, because the addition of new ones will multiply the number of features used during the project analysis. The use of too many features can excessively complicate further analysis. The features were chosen because of their simplicity and widespread use in the statistics [35], [36]. This stage allows to get definite number of important features that can be used for characterization of any individual project. The utilization of the same method will unify all projects that will be used in the analysis. There are many possible features that can be used, such as *Skewness*, *Median*, *Kurtosis* or *Fluctuation*; but increased number of features can significantly complicate analysis, though these features can be potentially considered in the future work, when relation between different features that will be outlined here will be clearer.

Table 15: The set of considered features extracted from sampled projects

#	Abbreviation	Name	Equation
1	μ	Mean	$\mu=\bar{x}=\sum_{i=1}^{n} x_i$
2	σ^2	Variance	$\sigma^2=\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2$
3	σ	Standard deviation	$\sigma=\sqrt{\sigma^2}=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2}$

Because of high correlation between quantities number 2 and 3, we decided to use only the latter one in the further analysis. Thus, only the **Mean** and **Standard Deviation** (STD) will be further used.

3.6.2 Description of extracted features in relation to categories

In regards to discussion given in sections 3.5.1, 3.5.2 and 3.6.1, for each collected project the following features were extracted, characterized in the table 16. In order to distinguish between different categories, the following notations are used: B – **Bugs**, F – **Features**, D – **Documents** and O – **Others** for the column "Applicability to the category". Statistical quantities, chosen in the section 3.6.1, will be calculated across all corresponding issue characteristics for four categories separately.

The rows 1 – 22 from the table 16 are used for four issue categories separately. Thus, overall number of obtained features will be multiplied by factor of 4. In such a way, we will have 88 distinct features. The rows 23 – 27 were used only once and were related to **Other** category. They provide 5 additional features. As it was mentioned earlier in section 3.5.1, the issue division between categories was done based on the label's content.

The rows 28 – 29 give some additional measurements about the project performance. The row 30 provides information on size of the collected project and gives the number of used issues in the project. The last row of the table is used for the project identification. The number of used issues in the project and project identification number are not used during the training stage.

The total number of different features obtained from each project is equal to 97. Not all of them will be explicitly used during the analysis. For example, irrelevant and weakly relevant features will be subject to removal from the feature vector. This topic will be discussed in the next section of the work. In addition, some further consideration will be needed with relation to fields that have missing values.

Table 16: The descriptions and statistical quantities used during the feature extraction from every project

#	Statistical quantity	Value description	Applicability to the category
1	Mean	The length of assigned issues in the project	B, F, D, O
2	STD	The length of assigned issues in the project	B, F, D, O
3	Mean	The number of times issues were reopened in	B, F, D, O

#	Statistical quantity	Value description	Applicability to the category
		the project	
4	Mean	The length of reopened state for issues in the project	B, F, D, O
5	STD	The length of reopened state for issues in the project	B, F, D, O
6	Mean	The number of comments in issues in the project	B, F, D, O
7	STD	The number of comments in issues in the project	B, F, D, O
8	Mean	The number of assigned issues in the project	B, F, D, O
9	Mean	The number of duplicated issues in the project	B, F, D, O
10	Mean	The number of won’tfix issues in the project	B, F, D, O
11	Mean	The number of people, who participated in the work on issue in the project	B, F, D, O
12	STD	The number of people, who participated in the work on issue in the project	B, F, D, O
13	Mean	Response time before the issue was created and first label was assigned in the project	B, F, D, O
14	STD	Response time before the issue was created and first label was assigned in the project	B, F, D, O
15	Mean	The number of commits for issues in the project	B, F, D, O
16	STD	The number of commits for issues in the project	B, F, D, O
17	Mean	The number of events for issues in the project	B, F, D, O
18	STD	The number of events for issues in the project	B, F, D, O
19	Mean	The number of issues in each of the category	B, F, D, O
20	Mean	The number of closed issues in each of the category in the project	B, F, D, O
21	Mean	The length of unassigned issues in the project	B, F, D, O
22	STD	The length of unassigned issues in the project	B, F, D, O
23	Mean	The length of unlabeled issues in the project	O
24	STD	The length of unlabeled issues in the project	O
25	Mean	The length of labeled issues in the project	O
26	STD	The length of labeled issues in the project	O
27	Mean	The number of labeled issues	O

#	Statistical quantity	Value description	Applicability to the category
28	Mean	The number of commits per user who participated in the project	Users
29	Mean	The merged “Pull requests” in the project	Pull requests
30	N/A	The total number of issues in the project	N/A
31	N/A	The identification number of the project in the database where collection was performed	N/A

3.7 The feature selection

Without taking into consideration missing values obtained after data extraction step, we found that few features showed quite small variance across all collected samples. According to [37], threshold by variance is a simple approach in feature selection. It allows to remove all features whose variance are equal or less of some threshold. The excluded features were not used or were used occasionally in small number of projects during the project development and/or supporting stages. They can be excluded from further analysis because they will not provide much relevant information. However, they can contribute valuable information in later work, when relations in the data can provide strong need for some of excluded features.

Table 17 shows variance values for all features used in the project in ascending order. The table 17 shows that feature numbers 3, 9, 10, 23, 29, 30, 43, 49, 50, 63, 69 and 70 have less than 0.012 variance (they are highlighted with gray background in the table 17). We decided to use the value of 0.012 for feature variance as a threshold for feature to be selected for the use in further analysis. The features 4, 5, 24, 25, 44, 45, 64 and 65 are larger than 0.012, but they have strong relations to the excluded features, number 3, 23, 43 and 63; thus, they were also excluded from consideration (these features are highlighted with dark gray background in the table 17). In such a way, the number of excluded features is equal to 20 and the number of total features that were used in the training was reduced from 95 to 75.

Generally, omitted features were related to rows numbers 3, 9 and 10 in the table 16. In addition, the feature number 3 has strong relations to rows number 4 and 5. As it was mentioned before, these features were seldomly used by GitHub users. For example, we found that “*won’tfix*” and “*duplicate*” issues were not used in most of the collected projects. Moreover, in these projects, where they were used, we saw just few examples with these issue labels. This factor can contribute to insignificant value that can be obtained from these features.

Table 17: The feature variances between the data, collected from all projects

#	Variance	#	Variance	#	Variance	#	Variance	#	Variance
50	7.49E-05*	20	7,01E-02	36	2,13E+00	24	8,84E+02	88	5,06E+04
30	1,94E-04	28	7,18E-02	66	2,37E+00	65	1,09E+03	14	5,16E+04
10	2,47E-04	8	7,73E-02	16	2,76E+00	4	1,32E+03	73	8,05E+04
9	2,63E-04	48	8,35E-02	77	3,84E+00	45	1,99E+03	13	8,49E+04
69	7,44E-04	40	8,55E-02	67	7,62E+00	5	5,00E+03	85	8,91E+04
70	1,00E-03	60	9,33E-02	37	9,94E+00	25	5,06E+03	21	9,64E+04
63	1,20E-03	75	1,13E-01	26	1,05E+01	94	7,29E+03	1	1,02E+05
29	2,31E-03	55	2,95E-01	46	1,13E+01	42	2,46E+04	87	1,11E+05
43	3,40E-03	71	3,67E-01	6	1,13E+01	54	2,98E+04	83	1,13E+05
49	3,86E-03	35	4,24E-01	17	1,22E+01	92	3,25E+04	33	1,26E+05
23	4,91E-03	31	6,53E-01	7	1,34E+01	84	3,27E+04	89	1,43E+05
59	7,51E-03	15	7,51E-01	57	1,93E+01	2	3,42E+04	53	1,58E+05
3	1,17E-02	11	7,51E-01	47	1,94E+01	22	3,88E+04	61	1,59E+05
19	1,28E-02	76	1,05E+00	18	2,19E+01	82	3,89E+04	91	1,73E+05
39	1,60E-02	32	1,18E+00	27	2,64E+01	90	4,03E+04	41	2,25E+05
68	2,08E-02	52	1,23E+00	78	2,84E+01	62	4,13E+04		
93	4,15E-02	56	1,59E+00	38	3,02E+01	74	4,25E+04		
80	5,61E-02	51	1,71E+00	58	3,20E+01	86	4,43E+04		
79	5,68E-02	12	1,80E+00	64	7.81E+01**	81	4,63E+04		
95	6,73E-02	72	2,02E+00	44	2,92E+02	34	4,85E+04		

* - features with gray background are excluded, because their variance is less than considered threshold of 0.012

** - features with dark gray background are excluded, because they are strongly related to features with small variance (i.e. less than 0.012)

3.8 Data cleaning

After feature selection, discussed in the section 3.7, we obtained data that contains features with *missing values* for **Bug**, **Feature** or **Document** related categories. The *missing values* are the values that were not possible to obtain during the data transformation due to different reasons. For example, the team used GitHub only as version control system without its project management extension (category division was not possible, because issues did not have labels), or the team used another non-English language for communication purposes, or different nonstandard label names have been used for issues, etc.

In order to improve this situation, the linear regression was used; it was trained on features from projects that did not have missing values in one of three categories. As predictors were used the values from **Other** category and as responses values from **Bugs**, **Features** or **Document** categories. The idea behind such approach can be outlined as follows: the history of the **Other** category with high probability influences other categories. The issue histories from the **Other** category can tell important statistical information from project history. Moreover, without availability of any other statistical data, this will be the best way to follow.

During further analysis, we found that missing fields occurred frequently in the new projects, where the total number of created issues is less than 35. To decrease influence of such data samples, we decided to exclude new projects from fuhrer consideration, although these projects can be safely used later for analysis on the trained model. Teuvo Kohonen showed that SOM is very efficient in dealing with missing values in such cases [25]. This step left just 230 samples in our training set.

3.9 Data normalization

In our work we used normalized project values. Normalization was performed before training of Self-Organizing Map. Data normalization is an important step used in data analysis. Usually, different features can have different scales or measurement units; sometimes it is hard to pinpoint the factor of importance for features in a data set. For example, expression of the feature in smaller scale will lead to wider range for this particular feature, in such a way, this situation increases the weight of the feature perceived by a learning algorithm [25].

When the factor of importance for different features is not known, data can be normalized in order to avoid the influence introduced by the choice of measurement units or different scales. We can transform the data to fall within smaller range such as [-1; 1] or [0; 1]. After transformation, all features will be equally important for the

training algorithm. For example, this transformation can be performed with the help of equation (10), called min-max normalization:

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\,max_a - new\,min_a) + new\,min_A \quad , \tag{10}$$

where min_A and max_A are the minimum and maximum value for the feature *A*. Min-max normalization performs a linear transformation of original value v_i of feature *A* to v_i' in the new range $[new\,min_A; new\,max_A]$. This technique preserves the relationships among the original data, but will encounter an "out-of-bounds" error if a future input case for normalization will fall outside of the original data range that was used for feature *A* [19, p113-115].

3.10 Summary

In this chapter we presented data model and data collection techniques. We analyzed raw data obtained from GitHub repositories. Based on labels content we found four different categories, and formed feature vectors that unified all collected projects. Features were divided on two groups. Features that were not frequently used were excluded; remaining features were cleaned and normalized for better performance.

The work done gives ground for the next stage of our research. In the next chapter, we will use obtained results for model training. Since we do not have any target results and knowledge about obtained data, we choose Self-Organizing Maps as this method is capable for unsupervised learning.

4. THE DATA ANALYSIS

In the beginning of this chapter, we formulate two hypotheses based on statistical data. Then, we proceed towards the description of training for self-organizing map. Later, we will outline different areas of the trained map and will evaluate the differences between statistical characteristics obtained across them. Near the end we will check outlined area performance in four categories that we discussed in the section 3.5.1. In the final part of the chapter, we will look at obtained results and possibilities to use the final model in the performance evaluation across software engineering projects deployed on the GitHub.

4.1 The hypotheses

4.1.1 Hypotheses formulation

We were interested to study how certain factors from the issue history influence issue duration. We formulated two hypotheses that check two properties, that are typically used in the issue work-flow: (1) the presence of assigned users, and (2) the presence of labels. Based on these properties, we formulated the following hypotheses:

I. Issues that had an assigned user will be shorter in length than unassigned ones.

II. Labeled issues will be shorter in length than unlabeled ones (this is only applicable for **Other** issue category, as, **Bugs**, **Features** or **Documents** have only labeled issues by default).

To verify these hypotheses, we will use statistical analysis on data collected from the GitHub.

4.1.2 Evaluation of the hypotheses from statistical data

We extracted mean length and standard deviation of the issues from our sample set. The results are shown in the tables 18 and 19. In these tables, the mean and standard deviation values of issue lengths were shown based on certain facts, such as:

- whether the issues in the categories are assigned or left unassigned throughout its lifetime in relation to the first hypothesis

- whether the issues are labeled or left unlabeled throughout its lifetime in relation to the second hypothesis.

For this analysis we did not apply any modifications to the initially obtained GitHub data. The features with missing values were omitted from the consideration. During the project analysis we found two particular projects with non-standard management habits (project identification numbers 13 and 14 in our database). The main problem in these project is that issues were reopened many times without any further changes to the issue statuses. Our script described in section 3.5.2 takes last reopened date and considers this date as the final issue closing date. Since these two projects are quite large, in the sense of number of created issues and have many assigned issues in **Bug** and **Feature** categories, this behaviour drastically modifies statistical results. We decided to exclude these projects from the hypotheses evaluation.

In the table 18, the data received for the evaluation of the first hypothesis is shown for four different categories: **Bugs**, **Features**, **Documents** and **Others**. Each category has columns for the mean and standard deviation of issue length calculated across all issues of certain category for assigned and unassigned issues separately.

Table 19 contains the data for the evaluation of the second hypothesis. In this table, relationship between issue lengths and presence or absence of labels in the issues throughout its lifetime was analyzed with relation to the **Other** category. Only the **Other** category contains unlabeled issues.

Table 18: The summary of statistical data obtained for the first hypothesis

	Assigned Bugs	**Unassigned Bugs**	**Assigned Features**	**Unassigned Features**
Mean length of issues	87,51	194.09	197,98	296,78
Standard deviation of issue lengths	198,81	421.45	287,42	451,34
	Assigned Docs	**Unassigned Docs**	**Assigned Others**	**Unassigned Others**
Mean length of issues	271,76	127,66	444,98	211,14
Standard deviation of issue lengths	437,87	340,16	654,60	469,45

Table 19: The summary of statistical data received for the second hypothesis

	Labeled Others	**Unlabeled Others**
Mean length of issues	164,09	258,10
Standard deviation of issue lengths	319,29	545.20

From table 18 we see that standard deviation values are bigger than mean values for all categories. For example, for the assigned **bug** and **feature** categories standard deviation values are bigger near 2.3 and 1.5 times than corresponding mean values. We know that duration lengths cannot be less than zero. This means that our issue duration distributions are not normal distributions and they are skewed toward the right side from mean values. Comparison between standard deviations values for assigned and unassigned **bug** category shows that unassigned standard deviation is longer 2.12 times than for assigned issues (1.57 for the **feature** category). This means that most lengths for unassigned issues skewed even further towards the right direction from mean values. We can assume that assigned issues have considerably shorter mean durations in time for first two categories with exceptions in third and fourth categories (**Document** and **Other**). On the other hand, document and other related issues were almost 100 per cent longer in the cases, where somebody was assigned for the work. The first two categories are the most important for the project functionality. The hypothesis 1 implies that for the issue to be resolved quicker in the **Bug** and **Feature** categories, project manager has to assign the person, who will be responsible for the work on the issue. Moreover, in this case not only the length will be shorter, but variance will be decreased, thus, the interval between the time an issue was opened and closed will be more predictable.

The data from the table 19 supports the second hypothesis for labeled and unlabeled issues. It shows that the issue length will be shorter in case if issue has label. The difference between length of labeled and unlabeled issues is near 57 per cent. As well as in previous case in the **Bug** and **Feature** categories, we see that variance of the length for labeled issues is significantly smaller (almost 71 per cent) with comparison to unlabeled issues.

Although we cannot state that hypotheses hold in all of the cases, we see that in our data strong relationship is present between different features. We showed that presence of some events, such as developer assignation or labeling, have some correlation with the length of an issue. At the same time, smaller variance allows to close an issues in foreseeable time. This fact can help the project manager to schedule the development and expect certain output from the team.

Finally, we can state that labeling and user assignation are good practices that can be successfully used in the GitHub work-flow. These practices can reduce the time needed for an issue to be resolved and improve project performance. Potentially, they help to clarify development process for all participated parties such as developers and users of software product.

4.2 The SOM training

In our work we used a Matlab environment and its Neural Network Toolbox for Self-Organizing Map training. Matlab provides an easy to use framework with possibility to utilize many different parameters for the training algorithms. The flexibility in changing of different parameters allows to improve performance of an output model and to fine-tune the model to fit our data in the best possible way. The toolbox supports many machine learning algorithms and has broad acceptance in the academia. The Neural Network Toolbox provides functions that are capable to visualize neural networks and simplify difficulties that arise in freedom of choice for the selection of final output model.

The training was based on 230 data samples produced by the data preparation step (section 3.5). After experimenting with different sizes of the lattice, we found that the best results is achieved with map of 9×7 neurons. Thus, the total number of used neurons in the SOM network is equal to 63 units.

For smaller sizes of trained maps, the number of training samples was insufficient. Training set did not allow to receive a good hit distributions for all samples in smaller trained maps. In this cases, areas with different characteristic were very close together and boundaries of these areas were pretty vague. On the contrary, the bigger map sizes resulted in dispersed and twisted hits, where some neurons were left without presence of any single hit. Such areas are harder to analyze, these neurons, that are laying close to boundaries did not receive any hits from our training sample set, and most of characteristics of such areas will be harder to predict in the future. In view of that, we decided to keep the size map of 9×7 neurons.

4.3 The resulting SOM and analysis of different areas, present in the map

Figure 17 shows the distribution of training sample hits to the neurons in our map. As it was mentioned above, the size of map is 9×7 neurons and we use hexagonal lattice, as proposed earlier in the section 2.10.6 by Teuvo Kohonen. The neurons were counted from 0 for the first bottom row, the second row has displacement of 1 in horizontal axis and the third row begins again from 0 and etc. The smaller blue inner hexagons inside

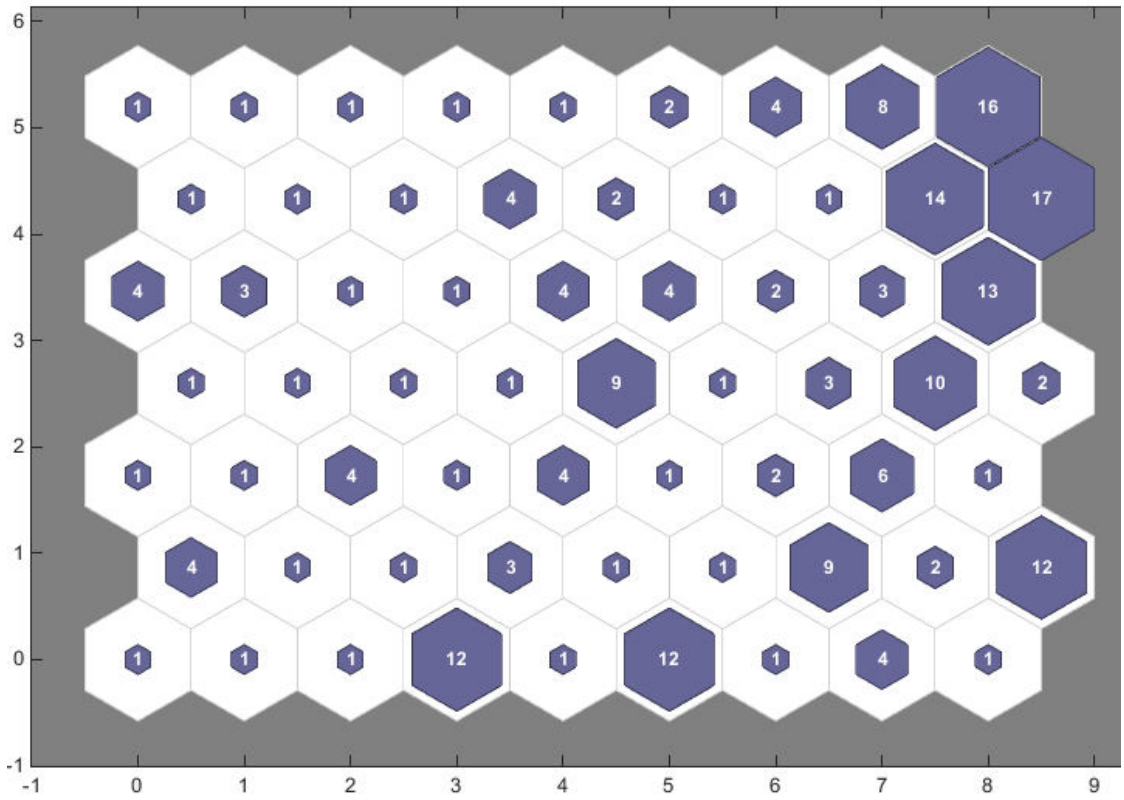

Figure 17: The self-organizing map sample hits received in the output model

neurons visually depict the number of hits received by the neurons after the training and are used for convenience reasons. The exact numbers of hits are shown by the white numbers in the center of each hexagon. The total sum of hits across all neurons in the map is equal to 230. This number corresponds to the number of samples that were used during the training stage, these samples were obtained during data cleaning step, in the section 3.8.

As it can be seen from the Fig. 17, most of the sample hits were on the right side of the map with slight displacement towards the upper right corner. The neurons from this corner had more hits than any other area in the whole map. The map did not have any empty neurons (without single hit). This fact allows us to analyze characteristic of each neuron with current training set.

After visual observation and analysis of different project characteristics and the corresponding distribution of hits in the neurons, we divided the map to nine distinct areas. We will inspect the projects characteristic for each of these areas. This method allows us simplify the analysis, because the number of subjects for the analysis is smaller. However, different neurons in one area represent slightly different characteristics and projects that were arranged in neighboring neurons observe gradual change in behaviour. The method allows us to start the analysis and decrease the number

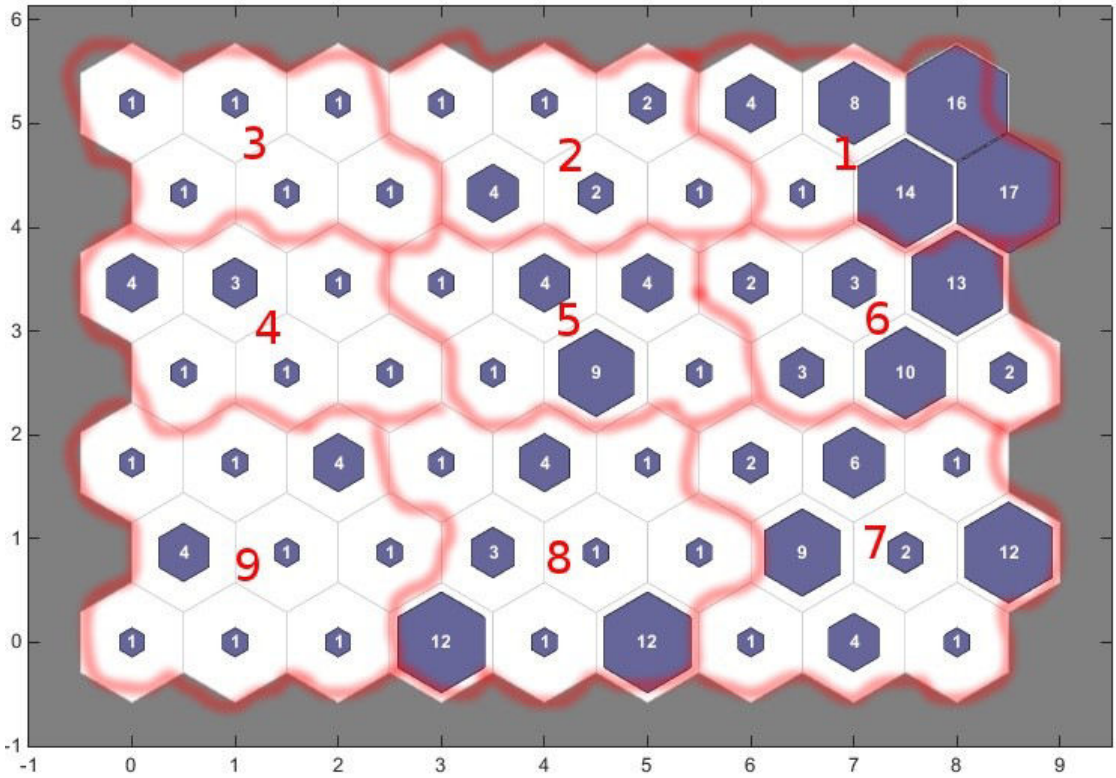

Figure 18: The outline scheme and enumeration for nine different areas that were analyzed

of subjects to the number we can deal with. In later stage of research, we can analyze gradual differences in neighboring neurons, when relation between these areas will be clearer. In the Fig. 18, the boundaries and numbers for these 9 different areas are outlined.

4.4 The characteristics of the areas

All features can be divided between two distinct groups: the first group combines features that are related to one of the **Bug**, **Features**, **Document** or **Other** categories and the second group combines additional features that do not depend on them. We will study results obtained for these groups separately.

4.4.1 The analysis of category related features

We analyzed the differences in the feature characteristics in all areas shown in Fig. 18. For this task, four tables with statistical characteristics were formed, for each of the four categories; each of the category was analyzed separately. The values of these tables were the mean values for each of the project characteristics calculated across all projects

that fall inside one area. These mean values are shown in the tables 21-24. The first column "#" is the index of the feature in reduced sample vector (after the feature selection stage, section 3.7). The "*Reference number*" (used as reference during analysis in all categories), measurement units and descriptions for tables 21-24 are shown in the table 20. For every number in column "*Number in the category*" in tables 21-24 corresponding description in the table 20 can be found with identical number from the column "*Reference number*". In the following text, we will use inter-category numbers (column "*Reference number*" from table 20) for the feature descriptions during the analysis in each of four categories.

Table 20: The joint description table with measurement units for the tables 21-24

Reference number	**Measurement units**	**Description**
1	days	The mean of length for assigned issues
2	days	The standard deviation of length for assigned issues
3	number	The mean of number comments used in issues in the category
4	number	The standard deviation of number comments in issues in the category
5	number	The mean of number assigned issues in the category
6	number	The mean of number people, who participated in the work on issues for the category
7	number	The standard deviation of number people, who participated in the work on issues
8	days	The mean of response time, before the issues was created and first label was assigned
9	days	The standard deviation of response time, before the issue was created and first label was assigned
10	number	The mean of number commits used in issues
11	number	The standard deviation of number commits for issues
12	number	The mean of number events for issues
13	number	The standard deviation of number events for issues
14	number	The mean of number of issues in the category to all issues used in the project
15	number	The mean of number closed issues in the category
16	days	The mean of length for unassigned issues
17	days	The standard deviation of length for unassigned issues

*Table 21: Feature characteristics for the **bug** category related issues received for 9 areas*

#	Reference number	Area 1	Area 2	Area 3	Area 4	Area 5	Area 6	Area 7	Area 8	Area 9
1	1	60.03	151.51	1207	610.07	198.02	111.17	75	212.6	351.09
2	2	36	104.92	485.39	302.12	148.14	125.86	74.18	144.56	129.94
3	3	4.37	3.61	4.26	4.45	3.61	5.29	4.02	3.83	4.12
4	4	4.21	2.27	3.98	4.93	3.93	5.49	3.55	3.69	3.23
5	5	0.26	0.17	0.34	0.09	0.15	0.29	0.21	0.17	0.17
6	6	1.99	1.77	1.85	2.11	1.73	2.34	1.82	1.84	1.8
7	7	0.9	0.6	0.84	1.98	0.92	1.69	0.85	0.97	1.1
8	8	13.72	59.71	961.37	796.6	185.78	39.4	62.03	213.02	588.89
9	9	20.89	92.37	669.83	686.32	280.11	83.62	89.49	210.38	457.24
10	10	0.69	0.61	0.34	0.92	0.84	0.75	0.63	0.72	0.42
11	11	0.86	1.06	0.62	1.48	1.36	1.32	0.93	1.37	0.72
12	12	5.91	4.38	4.71	4.29	4.52	6.78	5.1	4.45	3.72
13	13	4.28	2.53	2.7	4.23	3.82	6.15	3.79	3.76	2.67
14	14	0.13	0.08	0.25	0.1	0.09	0.16	0.11	0.12	0.12
15	15	0.82	0.8	0.33	0.55	0.68	0.78	0.85	0.74	0.6
67	16	64.62	172.53	1495.92	910.97	313.06	141.78	126.9	307.46	693.53
68	17	59.51	94.67	740.74	654.74	299.39	138.07	129.03	239.01	483.09

Based on statistical data shown in the table 21, we can see that the projects in area 7 did not have much people participated in work on issues; they did not have many assigned issues either. At the same time, in these projects we see good self-organization skills; as developers were capable to achieve very good performance in the terms of issue length and its variance for the **bug** related category. Projects from area 1 were the shortest ones; the team closed bug issues on average within 2 months. The projects, from areas 2, 6 and 7 also showed good performance in this category for assigned and unassigned issues.

We found certain patterns for the **bug** category. Considering the length of the assigned issues and its standard deviation: for feature numbers 1 and 2, the smallest values were from area 1 with sharp grow in the areas 2 and 3; in the area number 3 we observe longest length duration across whole map. At the same time, the grow of these parameters were moderate towards area 6 with small decrease in the area 7. The same can be said about features 8 and 9 that reflect response time (between creation time of an issue and first labeling) and in features 16 and 17 (unassigned issue length). We can see that in some areas the labeling time was exceptionally long, with exception in 1, 2, 6 and 7. The labeling task can be carried straightforward by the team manager and would not take significant working time. In such a way, the labeling that carried after more than 2 weeks from the date an issue was opened can suggest serious problem in the team management.

The length of unassigned issues (feature numbers 16 and 17) showed a pattern that confirmed to hypothesis 1. The unassigned issues had longer length with comparison to assigned ones in all nine areas. The differences in the length between assigned and unassigned issues were smallest in the projects from area 1 (close to 7 per cent). In other areas, the difference was at least few dozens of per cent. This fact shows that teams, whose projects were selected by neurons from the first area, have good management and organization capabilities.

In all projects, approximately equal number of comments were used, there was no observation of any sharp fluctuations (features 3 and 4). The same can be said about the people participation (features 6 and 7). However, the issues where the number of participants was close to 2 persons were shorter in general. The teams that had more assigned issues performed better and closed bug issues faster (feature 5). Features 10 and 11 are related to the number of commits used per issue. We found that teams, where more commits were used, closed issues faster. However, values do not differ significantly in this case. In areas, where the number of commits per issue was less than 0.5 (areas 3 and 9), we see longest issue length. In well performed projects, the mean number of events per issue was more than 5 (features 12 and 13). In such projects people used GitHub often for communication, clarification, reporting and development purposes. The status of development was clearer for the team and for users who participated in the development in comparison to projects, where the number of events per issue was smaller.

The feature 14 (the ratio of bug issues to all opened issues of project), did not show any interesting information; the values were approximately the same: near 10 per cent of all issues used in projects. The projects, where the number of closed bug issues were higher, performed better; the issue lengths for this category were shorter (feature 15).

*Table 22: Feature characteristics for the **features** category related issues, obtained for nine areas*

#	Reference number	Area 1	Area 2	Area 3	Area 4	Area 5	Area 6	Area 7	Area 8	Area 9
16	1	109.6	332.76	1146.97	677.59	322.69	193.05	183.55	324.81	460.81
17	2	79.48	194.85	578.24	344.68	200.89	175.93	135.32	220.51	197.59
18	3	3.83	4.78	4.17	4.53	4.29	5.11	4.31	3.93	3.17
19	4	4.62	6.41	3.86	5.72	4.97	6.48	4.93	3.93	3.03
20	5	0.24	0.2	0.31	0.09	0.14	0.29	0.21	0.16	0.17
21	6	1.79	2.04	1.78	2.07	1.72	2.27	1.7	1.55	1.59
22	7	0.86	1.54	1	2.28	1.17	1.58	1	1.02	1.04
23	8	27.8	264.66	1225.94	1102.39	339.97	79.54	148.43	434.05	813.65
24	9	42.81	220.62	686.2	639.45	329.63	129.52	193.93	320.42	473.87
25	10	0.42	0.37	0.29	0.87	0.62	0.45	0.44	0.5	0.46
26	11	0.78	0.73	0.72	1.61	1.32	1.12	0.88	0.98	1.6
27	12	5.4	5.22	4.87	4.47	4.36	6.41	4.86	3.95	3.21
28	13	4.88	6.05	3.21	5.66	4.59	7.02	4.63	3.75	3.04
29	14	0.15	0.09	0.07	0.06	0.07	0.13	0.12	0.05	0.07
30	15	0.69	0.57	0.41	0.38	0.51	0.56	0.6	0.48	0.42
69	16	106.14	353.07	1685.55	1212.63	463.88	246.02	292.65	540.38	919.39
70	17	97.52	246.12	700.51	580.06	322.1	179.26	246.82	306.61	408.69

The **feature** related category work-flow showed the same behaviour that we saw previously in the **bug** category, though the values have a different scale. We can summarize that our findings are mainly repeating the same statements from the **bug** category. As in previous case, the area 1 showed the best time performance. Similarly, the areas 6 and 7 had many good managed projects. The difference in lengths is significant between the **bug** and **feature** categories, thus the feature related issues were longer than **bug** related issues.

Regarding the features 1 and 2 (assigned issues), and 8 and 9 (response time), the projects showed the same properties as with previously mentioned category. Lengths of the unassigned issues (features 16 and 17) were longer than assigned ones. This supports the hypothesis 1. One small difference from previous case is in the area 1: it shows slightly smaller length in unassigned issues. It will be interesting to check behaviour for singular neurons of this area in later works. In addition, we noticed that all issue lengths were bigger than in the **bug** category. The number of comments do not differ significantly. We noticed that less than 4 comments were used in projects from areas with longest length (features 3 and 4). Feature 5 shows the ratio of assigned issues. Most of the projects have values close to 20 per cent, though some of areas with longest length have less than 17 per cent of assigned issues in the **other** category. As in the **bug** category, features 6 and 7 (people participation) did not provide any strong correlations with issue length, the same can be said regarding features 10 and 11 (number of commits used per issue). Similarly, in the features 12 and 13 (mean of events in the issues), when the number of events is close to 5, the length of the issues are shorter.

The most dynamical areas (in terms of issue length) were 1, 6 and 7; they have around 13 per cent ratio of issues from the **feature** category (feature 14). In these areas, the issues were closed faster and they had more events. On the other side, projects from other areas had about 7 per cent ratio of issues related to features. Areas 1, 2, 6, 7, where the rate of closed to open issues (feature 15) were close to 60 per cent, showed shorter length in assigned and unassigned issues. The smaller numbers of closed issues with comparison to **bug** category shows that projects are in active development. In addition, this situation can suggests that for the **bug** category priorities were higher, since projects had more closed issues.

*Table 23: Feature characteristics for the **document** category related issues, obtained for nine areas*

#	Reference number	Area 1	Area 2	Area 3	Area 4	Area 5	Area 6	Area 7	Area 8	Area 9
31	1	110.65	222.32	1169.43	527.79	248.76	106.49	119.89	198.8	340.71
32	2	54.61	185.98	432.89	314.43	188.7	99.93	104.15	171.56	278.4
33	3	5.01	4.72	4.56	4.4	5.07	5.66	4.21	4.43	3.88
34	4	3.69	2.89	2.78	4.01	6.36	5.46	3.33	3.63	2.74
35	5	0.18	0.12	0.24	0.03	0.1	0.2	0.14	0.06	0.04

#	Reference number	Area 1	Area 2	Area 3	Area 4	Area 5	Area 6	Area 7	Area 8	Area 9
36	6	2.19	2.13	1.79	1.91	1.95	2.33	1.73	1.76	1.8
37	7	0.84	0.73	0.61	1.27	1.28	1.49	0.64	0.85	0.8
38	8	24.87	397.92	1313	613.43	177.42	51.19	47.71	184.25	364.63
39	9	24.25	138.85	402.61	459.03	206.46	63.28	64	161.77	314.55
40	10	0.17	0.12	0.21	0.42	0.41	0.39	0.15	0.3	0.28
41	11	0.36	0.28	0.21	0.89	1.36	0.76	0.42	0.61	0.49
42	12	6.09	5.37	3.99	3.98	5.09	6.28	4.25	4.23	3.62
43	13	3.1	2.45	1.62	3.17	6.47	6.1	2.82	3.33	2.02
44	14	0.07	0.03	0.03	0.04	0.07	0.05	0.05	0.02	0.01
45	15	0.76	0.54	0.34	0.6	0.67	0.79	0.85	0.73	0.71
71	16	82.39	552.29	1489.14	632.57	264.3	137.92	87.59	228.08	396.72
72	17	57.31	133.14	495.06	447.02	241.78	103.96	72.64	173.45	313.39

The analysis of **document** related category was most difficult one in comparison to all other categories. We found that **document** category follows 2 previous ones, though it has some distinct characteristics: the length of unassigned issues can be shorter and the number of issues is low in the projects. We included in the **document** category wide range of issues related to documentation, such as questions from users or manual writing tasks. The characteristics of these issues differ significantly between each other. For example, a question can be answered by a single experienced user, unlike manual writing demanding opinion from many users, who are involved in the software development and application of software system.

Concerning the features 1 and 2, the length behaves in the same way as in the previous categories with one exception: the issue length in area 7 now is longer than 6. The features 8 and 9 follow **bug** and **feature** related categories. Unassigned issues (features 16 and 17) actually showed shorter lengths in some areas (1 and 7) with comparison to assigned ones. The reason behind this behaviour can be found in issue type for the **Document** related category. For instance, questions can be answered very fast, even before they can be assigned to someone, so for short issues managers did not use assignation. The number of comments (features 3 and 4) is an important factor used in work on this kind of issues. The comments are frequently used, to directly resolve an issue or to discuss document related matters. We found that relative number of

comments is higher than in any previous categories. However, further relation to issue length was not observed. For the number of assigned issues (feature 5) we did not notice any correlations with length of issues. The features 6 and 7 (number of people participated) did not provide any interesting information as in the previous two cases. In this category, smallest number of commits were used as many issues were related to questions or asked for a help (features 10 and 11). The features 12 and 13 that are related to number of events followed features 3 and 4 as most of the activity was done via comments. The relative number of the document related issues (feature 14) showed that this category included smallest number of issues; probably, developers have other means for communications, for example, via forums or chat channels. However, the number of closed issues were high in most of the projects. Feature 15 (number of closed issues) showed that developers prioritized this type of issues in general.

*Table 24: Feature characteristics for the **other** category related issues, obtained for nine areas*

#	Reference number	Area 1	Area 2	Area 3	Area 4	Area 5	Area 6	Area 7	Area 8	Area 9
63	1	42.5	245.25	992.11	215.77	178.09	79.16	31.46	55.69	229.65
64	2	34.15	101.81	439.81	198.19	216.34	111.69	42.83	52.04	80.5
48	3	3.29	3.95	3.31	3.51	3.79	3.54	3.17	2.81	2.39
49	4	4.04	4.73	4.42	4.64	4.67	4.99	3.9	3.43	2.86
50	5	0.09	0.1	0.24	0.01	0.05	0.13	0.06	0.04	0.02
51	6	1.53	1.61	1.53	1.66	1.55	1.67	1.39	1.24	1.32
52	7	1.43	1.19	1.05	1.85	1.15	1.6	0.97	0.97	1.14
53	8	33.46	217.36	1056.82	648.8	195.67	59.88	96.82	206	406.26
54	9	54.54	304.79	626.45	642.64	313.67	118.22	177.57	288.29	515.55
55	10	0.27	0.31	0.19	0.52	0.45	0.36	0.27	0.31	0.28
56	11	0.82	0.74	1.5	1.57	1.16	0.95	0.72	1.07	0.72
57	12	3.47	3.68	3.31	2.97	3.43	3.9	3.06	2.42	2.22
58	13	5.24	4.38	3.42	4.07	4.42	7.36	4.02	3.2	2.52
59	14	0.66	0.8	0.68	0.84	0.81	0.67	0.74	0.87	0.84
60	15	0.82	0.69	0.48	0.53	0.71	0.83	0.8	0.67	0.62
65	16	39.24	222.44	1244.69	649.73	213.87	79.14	98.81	213.64	410.31
66	17	61.69	308.9	757.95	644.99	316.4	133.74	174.59	292.17	516.64

The **other** category embraces all issues that were not included in the **bug**, **feature** or **document** related categories. The **other** category has supplementary issues that were used to guide the development process. The issues were short, they did not have many assigned users and the issues did not have many commits. As we see, the more issues were closed the more experienced developers has project that influence the length of an issues. In addition, most successful projects dispersed issues across many categories by labeling. In such a way, project teams have smaller number of issues in the **other** category. For example, in the most effective area 1 we see the smallest mean value of the feature number 14. This feature reflects the number of issues in the **other** category divided by overall number of issues used in the project. In 1 area its value is equal to 66 per cent, thus, the rest 34 per cent belong to different categories of the projects.

The issues from this category were the shortest ones across all categories that we analyzed in terms of unassigned issue length the features 16 and 17 (as they constitute the majority of issues). This category can incorporate all issues from a project, for example, when non-standard label names were used or developers used different non-English languages. In such a way, the **bug**, **features** and/or **document** related categories will be left empty. During the model training and analysis they will be filled with the help of linear regression (as was discussed earlier in the section 3.5).

The length of the assigned issues (features 1 and 2), response time (features 8 and 9) and unassigned issues length (features 16 and 7), observes the same behaviour as in previously analyzed categories. The hypotheses 1 holds in most of the areas. As in earlier cases, the number of comments (features 3 and 4) do not give any interesting relations. The feature 5 shows that number of assigned issues in **other** category is very low and on average is less than 10 per cent. Most probably, this category was not considered as an important one by the development teams. The same conclusions can be observed under examination of features 6 and 7 that shows number of people, participated in work. These values have small variance across all areas that we used. Under examination of number of commits (features 10 and 11) and number of events (features 12 and 13) we did not find any interesting patterns. The number of closed issues follows the length of issues. For example, the shorter length of unassigned or assigned issues the higher rate of closed issues in project. The category includes most of the issues used in projects as showed by feature 14.

4.4.2 The analysis of supplementary features

The table 26 contains seven additional features that do not depend on any of the categories; they were collected supplementary and were used for additional performance tests. As in previous section table 25 contains joint descriptions and measurement units for corresponding values shown in table 26. Column "*Feature Number"* gives direct

feature numbers from reduced sample vector (after the feature selection stage described in the section 3.7)

Table 25: The direct reference number and measurement units for features and its descriptions used for additional features in the project

Feature Number	**Measurement unit**	**Description**
46	days	The mean of length for labeled issues in the project
47	days	The standard deviation of length for labeled issues in the project
61	days	The mean of length for unlabeled issues in the project
62	days	The standard deviation of length for unlabeled issues in the project
73	number	The mean of number labeled issues to unlabeled
74	number	The mean number of commits per user who participated in the project
75	number	The mean value of “Pull requests” in the project that were merged

Table 26: The direct number of feature and its corresponding value used for additional features in the project

Feature Number	**Area 1**	**Area 2**	**Area 3**	**Area 4**	**Area 5**	**Area 6**	**Area 7**	**Area 8**	**Area 9**
46	47.23	361.94	1471.9	417.42	387.46	175.61	60.12	60.28	205.67
47	39.24	322.43	572.77	368.61	326.5	160.9	80.51	63.88	226.84
61	34.94	184.5	602.79	634.98	180.53	57.18	101.03	204.36	396.27
62	55.02	263.81	488.36	606.35	297.53	103.96	174.15	281.85	507.92
73	0.19	0.21	0.36	0.18	0.21	0.21	0.14	0.07	0.07
74	31.07	19.41	100.06	18.09	45.15	55.31	40.52	42.2	24.67
75	0.65	0.68	0.3	0.38	0.65	0.6	0.63	0.62	0.59

The analysis of the supplementary features is mostly related to the hypothesis 2. We see that it holds only in small fraction of areas: 7, 8 and 9. In addition, it was mentioned that in these areas the relative rate of labeled to all issues is very small (feature 73), it is around 14 per cent and less, whereas another area showed more then 18 per cent. The projects from areas 1, 6 and 7 were again the shortest ones with the mean length less than 200 days. In most of the areas, utilization of “Pull requests” was perfect, more than

60 per cent of them were merged in the projects. Alternatively, most ineffective projects and projects that had longest issues length had less than 40 per cent of merged issues rate. The feature number 74 did not provided many relevant information, most probably, because we did not make any distinction between the developers, who are capable to create commits, and an ordinary users. The latter cannot produce many commits, but mainly discuss issues. For this reason, the parameter got spread between both categories of users altogether. In such a way, the results were mainly influenced by the factor of user activity in the project and not by the number of commits that were made by developers.

Based on analysis of the features 46 and 47, we see that labels were created mainly for issues that were time consuming. On the other hand, for the shortest issues that do not demand much working time labels were omitted. This is one of the reasons why the hypothesis 2 does not hold in this case.

4.4.3 The summary of results

During our analysis we saw that projects from different areas significantly differ in characteristic between each other. This fact suggested that division of projects performed by Self-Organizing Map was successful.

We saw that in tables 21-24 and 26 that mean and standard deviation features obtained from the same parameters have quite big relative values. For example, in the table 21 that is related to the **Bug** category, features number 1 and 2 (mean and standard deviation length values for assigned bug issues) in areas 6 and 7, or features number 16 and 17 (mean and standard deviation length values for unassigned bug issues) in areas 1, 5, 6, 7 and 8 have almost equal values. This can suggest significant deviation in the projects that fell in one or another area. However, if we will compare adjacent areas we will see that same features from projects of these areas has different orders of magnitude. For example, in the table 21 values for features number 1 and 2 differ drastically between neighboring areas 1 and 2, 1 and 6, 6 and 7 and etc. Thus, these areas are forming different cluster of samples. In addition, it is worth to remember that neurons are firing on the basis of discriminant function calculations. These calculations are performed across all features of a sample, as was discussed earlier in section 2.10.1. Different features will have different factor of importance for distinct areas. The SOM output is calculated on the basis of full feature vector, not one particular feature or even subset of features from any individual sample. In this way, all features of sample will influence output of the SOM together. This factor can explain some deviation obtained in projects characteristics that were put in one area.

The **Other** category has the issues with shortest length in comparison to all other categories. This fact can suggest that issues from this category in general require less effort for solving. We found that this category has the biggest amount of issues used during project development. The smallest number of issues from this category was found in the area 1 (table 24), where projects had near 66 per cent of total issues that were used during development. Thus, we can say that issues belonging to projects from area 1 were most dispersed across different categories as **Bug**, **Feature** and **Document** related ones. We found that issues from projects of this area has smallest length we can conclude that labeling can clarify meaning and purpose for issues and ease management process.

Finally, we conclude that the behaviour of the projects in all categories are heavily connected between each-other. For example, projects from area 1 show similar behaviour in length for four found categories. The same behaviour we can observe from other areas, but in different scale. This fact verifies our assumption about mutual dependence inherited in data obtained for the **Bug**, **Feature**, **Document** and the **Other** categories that was made in the earlier section (3.8).

4.5 The projects quality, intermediate results

We outlined performance of projects on self-organizing maps, based on different factors that we considered before. The map shown in Fig. 19 is in draft stage that need to be verified later. As neurons supposed to reflect gradual change in sample characteristics

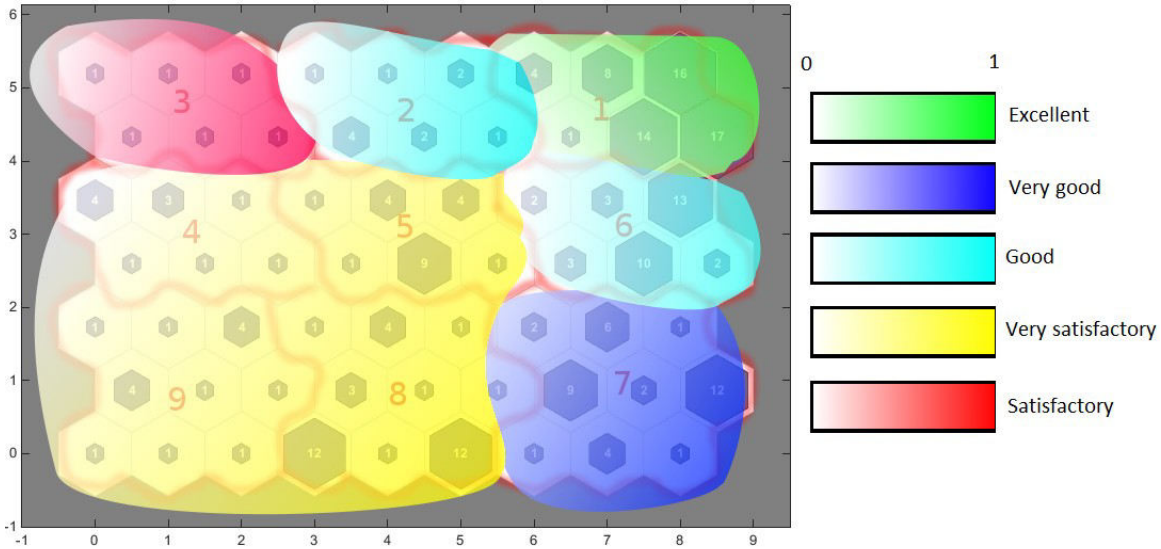

Figure 19: Intermediate region performance scores outline on the left side and corresponding performance measures, showed on the right side

we used gradient to highlight change in project quality that can be observed with transition from one to another regions.

The projects from area 1 showed the best performance in the analysis of all four categories and supplementary features we marked this area as the area with excellent project performance. The projects from area 7 show very short issue length, moreover, the issue lengths for the **bug** category were on the second place after the area 1. Thus, the teams from this area prioritized bug issues to any other issues types. As these issues are essential for user experience and project stability, this region was considered as having very good performance score. The areas 2 and 6 together formed another region where projects share similar characteristics. The projects that fall inside of this region showed good performance scores across all feature set. As consequence, projects from these areas took middle position in our rating. The area 3, clearly, showed worst performance based on all feature characteristics, across all four categories; the same can be said in regard to the additional features. The areas 4, 5, 8 and 9 took intermediate value between the area 3 from one side and areas 2 and 6 from another, thus, this region was assigned a satisfactory performance score with worst projects from region located in the area 4, as nearest to the region that has satisfactory scores.

5. EVALUATION

Based on results obtained in section 4, we see that the Self-Organizing Maps is a powerful tool that can help to perform data analysis. However, without any initial knowledge in the application area it is difficult to produce any feasible model suitable for the analysis. It is hard to analyze initial *raw* data collected from the GitHub repositories. The *raw* data is set of records consisting mainly from issues and all corresponding events with related description.

We produced the model that used natural GitHub work-flow. The model split all issues of projects to the different categories: **Bug**, **Feature**, **Document** and **Other**. Then we checked different interesting events used in the GitHub work-flow. With the help of the model we showed that we are able to extract relevant information from the collected data.

The Self-organizing maps are capable to arrange different samples based on similarity measures between different features of these samples. For example, we showed that on one side developers prioritize **feature** related issues, whereas, on the other side they resolve **bug** issues at first. We found that different projects have distinct behaviour based on project team habits and manner of work. We refer to the team as to the set of all participants, who made any contribution during the project development stage. The set includes developers and ordinary users, where the former can change source code, make commits or carry on project development and the latter can open an issues.

The individual project characteristics were encoded in a spacial location of neurons of our model during the training phase. All neurons were tuned in such a manner “to fire” in case of presentation of similar patterns. We found areas with the best and the worst evaluation scores in previous section of this work (4.5).

The difference in parameters, such as issue length, number of events and event type between four categories was considerable. We see different patterns showed by the same team during work-flow process in the **Bug**, **Feature**, **Document** and **Other** categories. We can conclude that category division was successful as it helped to highlight different patterns inside project teams and clarified actual priorities utilized by developers for different categories of issues. In most successful projects many issues were put in separate categories and the **Other** category has lowest number of issues of projects. In such a way labels clarify meaning for issues, improve cooperation inside the team and

show next important issues to start work. In this light the use of clear and easy to understand labels is essential for efficient results.

We found that some of the features did not provide any relevant information in current dimensionality during the analysis. For example, the number of commits or events did not vary significantly between neurons of the map. This information did not show any contribution to the length of assigned or unassigned issues and can be excluded from the current analytical model or can be incorporated in some other way in the future work.

To verify our results, we were intended to utilize one of independent systems, where users can leave a feedback about own experience and feature usability scores in relation to any software product of interest. However, we found that most of collected samples are libraries, additional modules, or small products that do not have any records in databases of such systems. On the other side, we have a big projects, for example, Node.js, it is an open-source, cross-platform run-time environment, that can be used for development of server-side Web applications. The project has few recent surveys, available in open access, however, the surveys are mostly relate toward technologies or discuss the project areas for which this software was utilized. These surveys do not include any references to the actual team performance and do not have any user experience scores for the product or alternatively they were intentionally omitted from an open access. Due to complication in application of this approach we had to leave it as a future work.

6. CONCLUSIONS AND FUTURE WORK

This thesis described the research and design of data collection, storage, model and evaluation systems as part of a non-intrusive automated system for evaluation of project performance. The motivation behind this thesis was the need for utilization of available software engineering data in process improvement in the software project management. The primary target is utilization of already existing data that was generated during the project development stage. We assumed that this data is an important source of information about software project work-flow. The GitHub repositories were utilized as a source of software engineering data. We defined additional limitations, to be applied on our system:

- The minimum human participation.
- The absence of the expert in the software engineering field during project evaluation.
- The possibility to include in the evaluation an additional systems that can be used in software engineering field.

The abundance of software engineering data in the open access has produced currently existing systems that can be used for project evaluation. However, these systems have restricted functionality as they: can work only with few projects, require trained user for evaluation, or suitable for evaluation only in a fraction of software engineering fields. These facts pose a challenges in wide-scale application of such systems in the real life. Advancements in the Artificial Intelligence and Machine Learning methods suggested us that any evaluation task can be automatized in a way to produce feasible results without any human participation, within short operational time and with reasonably low expenses. Moreover, such automated system can be trained using broad collection of samples, which can be difficult to evaluate manually for any particular individual or expert in the field.

To get a full understanding of the problem, the existed data collection model was thoroughly analyzed and the modified data model was proposed. New model simplifies data model design, improves storage space requirements and search speed in database. Based on issue labels content, we found three distinct issues categories, extracted statistical characteristics from collected data, and selected relevant features for the

analysis. As we worked with data without any available labeled responses we used Self-Organizing Maps. This method is capable to unsupervised learning. Received samples were distributed across the map of 63 neurons. We observed distinct behaviour in different areas of this map. These differences were used in the project evaluation part of the work.

During analysis stage, we found that some of the features do not show any influence to the length of issues and do not provide any relevant information; they can be omitted from the analysis. However, due to results obtained in the section 4.4 with features, related to duration between different events of an issue and an issue length, we propose to change dimensionality of such features in further work. For example, to explore the possible connections in relation to the issue length with the mean time between occurrence of commit events in issues.

Collected data allowed us to partially prove the first hypotheses and to prove the second one (section 4.1). Nonetheless, we must point out that the difference in the length that we found with relation to assignation and labeling can be due to the influence produced by some other factors, which were not taken into consideration. For example, assignation can be done for issues with shorter time needed for a work; more experienced users tend to assign themselves and others do not. In any circumstances such an easy and non-time consuming practices as user assignation and labeling can potentially clarify status of the development and decrease randomness in the team behaviour. We suggest to use them in the project management as early as possible.

The features that were obtained from collected data (section 3.6) were used for training of Self-Organizing Map. We found that characteristics obtained for projects in different regions of the SOM differ significantly between each other. We saw that in projects in region with shortest mean issue resolution time more issues were distributed between **Bugs**, **Features**, **Documents** categories than in any other region. In addition assigned issues showed better performance in heavier load issues, that demand more working time in **Bugs** and **Features** categories in most of the regions. We can conclude that management plays significant role in team efficiencies and can decrease issue resolution time and issue deviation length. For efficient management it is essential to apply user assignation for time consuming issues and label them with easily recognizable descriptions from earliest stage of development. Such practices ones employed may significantly improve team efficiency, software stability and in such a way user experience and satisfaction scores.

The future work includes validation of results, developing separate system for evaluation requests from the users, extending possibility to collect data from different software project management systems. The validation stage can be performed with the

help of user related information that was collected from the GitHub during data collection stage. This information includes, email, names and number of opened issues or created events by any particular user. During validation stage, a questionnaire can be sent via emails with intention to collect user grades with regards to project performance in the different areas of interest. The feedback from questionnaire can clarify common problems and mark corresponding areas from the SOM, where such problems are commonly present. In spite of previously mentioned limitations, our work can be used as a starting point for the development of more complex applications.

REFERENCES

[1] Wikipedia, “Conceptual schema definition.” [Online]. Available: https://en.wikipedia.org/wiki/Conceptual_schema. [Accessed: 11-Aug-2016].

[2] B. Boehm and R. Ross, “Theory-W Software Project Management: Principles and Examples,” *IEEE Trans. Softw. Eng.*, vol. 15, no. 7, pp. 902–916, 1989.

[3] M. Levesque, “Fundamental issues with open source software development (originally published in Volume 9, Number 4, April 2004),” *First Monday*, no. 2, Oct. 2005.

[4] M. K. Waaijer, *Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives (St. Amant, K. and Still, B., Eds.; 2007) [Book review]*, vol. 52, no. 1. 2009.

[5] “Github web page.” [Online]. Available: https://github.com/. [Accessed: 14-Jan-2016].

[6] “JIRA web page.” [Online]. Available: https://www.atlassian.com/software/jira. [Accessed: 14-Jan-2016].

[7] “Atlassian Open Source license request.” [Online]. Available: https://www.atlassian.com/software/views/open-source-license-request. [Accessed: 15-Jan-2016].

[8] TheLawDictionary.org, “Engineering data.” [Online]. Available: http://thelawdictionary.org/engineering-data/. [Accessed: 02-Feb-2016].

[9] A. Mattila, O. Sievi-Korte, A. Eteläaho, K. Kuusinen, M. Leppänen, and K. Systä, “Timeline Based Issue Management Data Visualization for Observing Software Projects.”

[10] O. Hylli, A. Mattila, and K. Syst, “Collecting Issue Management Data for Analysis with a Unified Model and API Descriptions,” in *14th Symposium on Programming Languages and Software Tools, At Tampere, Finland*, 2015, pp. 1–15.

[11] T. Lehtonen, V.-P. Eloranta, M. Leppanen, and E. Isohanni, “Visualizations as a Basis for Agile Software Process Improvement,” *Softw. Eng. Conf. (APSEC), 2013 20th Asia-Pacific*, vol. 1, no. November 2012, pp. 495–502, 2013.

[12] NASA, “NASA - Dryden Flight Research Center - News Room: News Releases: NASA NEURAL NETWORK PROJECT PASSES MILESTONE,” 2003. [Online]. Available:

http://www.nasa.gov/centers/dryden/news/NewsReleases/2003/03-49.html#.VrHFerJ97IV.

[13] M. F. Kader, “Neural Network-Based English Alphanumeric Character Recognition,” *Int. J. Comput. Sci. Eng. Appl.*, vol. 2, no. 4, pp. 1–10, Aug. 2012.

[14] P. Louridas and C. Ebert, “Machine Learning,” *IEEE Softw.*, vol. 33, no. 5, pp. 110 – 115, 2016.

[15] “GitLab.” [Online]. Available: https://about.gitlab.com/. [Accessed: 19-Jan-2016].

[16] “Bitbucket.” [Online]. Available: https://bitbucket.org. [Accessed: 19-Jan-2016].

[17] T. M. Mitchell, *Machine Learning*, no. 1. McGraw-Hill, 1997.

[18] A. O. Sykes, “An Introduction to Regression Analysis,” *Am. Stat.*, vol. 61, no. 1, pp. 101–101, 2007.

[19] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*, 3rd ed. 2012.

[20] J. Tohka and J. Niemi, “Feature Selection and Extraction Lecture slides SGN-41006,” 2015.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[22] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. PEARSON EDUCATION LIMITED, 1998.

[23] T. Kohonen, “Self-organized formation of topologically correct feature maps,” *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.

[24] G. Rickheit and I. Wachsmuth, *Situated Communication*. DE GRUYTER MOUTON, 2008.

[25] T. Kohonen, *Self-Organizing Maps*, 3rd ed., vol. 30. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[26] T. Kohonen, “The self-organizing map,” *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[27] M. Kantardzic, *Data Mining*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011.

[28] I. Skerrett, “Eclipse Community Survey 2014 Results,” 2014. [Online]. Available: https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/. [Accessed: 19-Nov-2015].

[29] Github, “The information about Github events.” [Online]. Available: https://developer.github.com/v3/issues/events/. [Accessed: 09-Mar-2016].

[30] A. Mattila, A. Luoto, H. Terho, O. Hylli, O. Sievi-korte, and K. Syst, “Unified Model for Software Engineering Data,” pp. 150–154, 2015.

[31] A. Mattila, M. Niemel, and A. Etel, “What Can be Revealed by Visualizing Software Engineering Data ? – a Case Study.”

[32] I. MongoDB, “MongoDB embedded data.” [Online]. Available: https://docs.mongodb.org/manual/core/data-model-design/#data-modeling-embedding. [Accessed: 01-Mar-2016].

[33] S. Allen, “SQL Performance Tuning using Indexes,” 2004. [Online]. Available: http://odetocode.com/articles/237.aspx. [Accessed: 01-Feb-2016].

[34] “About Firebird.” [Online]. Available: http://www.firebirdsql.org/en/about-firebird/. [Accessed: 02-Mar-2016].

[35] J. S. Lee, “Digital image enhancement and noise filtering by use of local statistics.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 2, no. 2, pp. 165–168, 1980.

[36] H. Huttunen, T. Manninen, J. P. Kauppi, and J. Tohka, “Mind reading with regularized multinomial logistic regression,” *Mach. Vis. Appl.*, vol. 24, no. 6, pp. 1311–1325, 2013.

[37] “Feature selection.” [Online]. Available: http://scikit-learn.org/stable/modules/feature_selection.html. [Accessed: 12-Aug-2016].

APPENDIX A: DATABASE TABLE CREATION

```
set sql dialect 3;
set names WIN1251;
connect 'C:\Users\Alexey\Dropbox\TUT\Thesis\DB\PARSERUM.fdb'
USER 'Fermiy' password 'DiscO36';
/*Domain creation for text fields*/
create domain titleSt varchar(60);
create domain usernm varchar(80);
create domain desctiptSt varchar(200);

/*Projects table*/
create table PROJECTS
(
PRJID integer not null,  /*Unique code used for projects*/
PROJECTNM desctiptSt not null,     /*name for project or address stri
PRSDATE timestamp not null, /*Date when project was saved from system
constraint PK_PROJECTS PRIMARY key (PRJID)
);
create generator GEN_PROJECTS;
create UNIQUE INDEX ID_PROJECTS on PROJECTS (PROJECTNM);

/*Tables for user data storage*/
create table USERS
(
USERID integer not null,   /*user id from outer system*/
USERNAME usernm,
USERLOGIN usernm not null,
EMAIL usernm,
constraint PK_USERS PRIMARY key (USERID)
);

/*Tables for label storage*/
create table LABELS
(
LBNM titleSt not null,   /*name of label from outer system without any
LBID integer not null,
LBCOUNT integer not null,
constraint PK_LABELS PRIMARY key (LBNM),
constraint UK_LABELS UNIQUE (LBID)
);
create generator GEN_LABELS;
create UNIQUE INDEX ID_LABELS on LABELS (LBID);


```

```
44 /*Used for storage of different event types*/
45 create table TYPES
46 (
47 TYPID integer not null,   /*name of label from outer system without a
48 TYPENM varchar(20) not null,
49 constraint PK_TYPES PRIMARY key (TYPID)
50 );
51 create generator GEN_TYPES;
52 create INDEX ID_TYPES on TYPES (TYPENM);
53
54
55 /*Used for storage issues of different projects*/
56 create table ISSUES
57 (
58 PRJID integer not null,   /*name of label from outer system without a
59 ISSUEID integer not null,   /*issue id from outer system*/
60 USERID integer not null,   /*user id from outer system*/
61 ISSTIME timestamp not null, /*time issue was created*/
62 ASSIGNID integer,   /*assigned user id from outer system*/
63 TYPEISS char(1) not null,   /*issue type '0'-pull requests, '1'-issue
64 DESCRIPT desctiptSt not null,
65 TITLE titleSt not null,
66 COMMENTS integer not null,    /*how many comments issue got*/
67
68 constraint PK_ISSUES PRIMARY key (PRJID,ISSUEID),
69 constraint UK_ISSUES UNIQUE (ISSUEID),
70 constraint FK1_ISSUES FOREIGN key (PRJID) REFERENCES PROJECTS (PRJID)
71     ON delete CASCADE
72     ON update CASCADE,
73 constraint FK2_ISSUES FOREIGN key (USERID) REFERENCES USERS (USERID)
74     ON delete CASCADE
75     ON update CASCADE,
76 constraint FK3_ISSUES FOREIGN key (ASSIGNID) REFERENCES USERS (USERID
77     ON delete CASCADE
78     ON update CASCADE,
79
80 constraint CH1_ISSUES CHECK (TYPEISS IN ('0','1'))
81 );
82 create INDEX ID1_ISSUES on ISSUES (TYPEISS);
83
84
85 /*Used for storage of labels used for issues*/
86 create table ISSLABELS
87 (
88 ID integer not null,        /*Artificial key*/
89 PRJID integer not null,   /*name of label from outer system without a
90 ISSUEID integer not null,   /*issue id from outer system*/
```

```
 91 LBID integer not null, /*label id*/
 92 constraint PK_ISSLABELS PRIMARY key (ID),
 93 constraint   FK1_ISSLABELS   FOREIGN   key   (PRJID,ISSUEID)   REFEREN
 94 (PRJID,ISSUEID)
 95     ON delete CASCADE
 96     ON update CASCADE,
 97 constraint FK2_ISSLABELS FOREIGN key (LBID) REFERENCES LABELS (LBID)
 98     ON delete CASCADE
 99     ON update CASCADE
100 );
101 create generator GEN_ISSLABELS;
102
103 /*Used for storage events of different projects*/
104 create table EVENTS
105 (
106 PRJID integer not null,   /*name of label from outer system without a
107 EVID integer not null,   /*event id from outer system*/
108 TYPID integer not null,   /*issue type index from TYPES*/
109 TYPENM computed by ((select TYPENM from TYPES where events.typid=type
110 REFISSUE integer,   /*reference to the issue that this event is relat
111 ACTORID integer,   /*actor id produced this event*/
112 EVTIME timestamp not null, /*time event was produced*/
113 DESCRIPT desctiptSt,
114
115 constraint PK_EVENTS PRIMARY key (PRJID,EVID),
116 constraint FK1_EVENTS FOREIGN key (PRJID) REFERENCES PROJECTS (PRJID)
117     ON delete CASCADE
118     ON update CASCADE,
119 constraint FK2_EVENTS FOREIGN key (TYPID) REFERENCES TYPES (TYPID)
120     ON delete CASCADE
121     ON update CASCADE,
122 constraint FK3_EVENTS FOREIGN key (ACTORID) REFERENCES USERS (USERID)
123     ON delete CASCADE
124     ON update CASCADE
125 );
126
127 /*Used for storage of commits*/
128 create table COMMITS
129 (
130 PRJID integer not null,   /*name of label from outer system without a
131 COMMID varchar(40) not null,   /*commit id from outer system*/
132 USERID integer,   /*user name id*/
133 COMMTIME timestamp not null, /*time event was produced*/
134 MSG desctiptSt not null,
135 COMMENTS integer not null,
136
137
```

```
138 constraint PK_COMMITS PRIMARY key (PRJID,COMMID),
139 constraint UK_COMMITS UNIQUE (COMMID),
140 constraint FK1_COMMITS FOREIGN key (PRJID) REFERENCES PROJECTS (PRJID
141     ON delete CASCADE
142     ON update CASCADE,
143 constraint FK2_COMMITS FOREIGN key (USERID) REFERENCES USERS (USERID)
144     ON delete CASCADE
145     ON update CASCADE
146
147 );
148
149 /*Used for storage parents of commits*/
150 create table COMMPARNT
151 (
152 ID integer not null,        /*Artificial key*/
153 PRJID integer not null,   /*name of label from outer system without a
154 COMMID varchar(40) not null,   /*commit id from outer system*/
155 PARENTS varchar(40) not null,   /*user name id*/
156 constraint PK_COMMPARNT PRIMARY key (ID)
157 );
158 create generator GEN_COMMPARNT;
159
160 COMMIT;
    EXIT;
```

APPENDIX B: PROJECTS COLLECTED FROM GITHUB

Project ID used in the database	Project name on the GitHub
13	dart-lang/sdk
14	phpmyadmin/phpmyadmin
15	neovim/neovim
16	nodejs/node
17	ViennaRSS/vienna-rss
18	alexgorbatchev/jquery-textext
19	vubinhduong91/backboneJS
20	chef-cookbooks/php
21	ThomasBurleson/angularjs-Quizzler
23	phayes/geoPHP
24	aashuagg/Bootstrap-Validator
25	fumzilla/backbonejs
26	LuvDaSun/angular-hal
27	chriso/klein.php
28	walu/phpbook
29	rails/jquery-rails
31	textmate/javascript.tmbundle
32	getsentry/raven-php
33	chrisboulton/php-resque
44	MPOS/php-mpos
45	oyejorge/less.php
46	wankdanker/node-odbc
47	xdenser/node-firebird-libfbclient
48	RubaXa/jquery.fileapi
49	Atmosphere/atmosphere-javascript
50	laravel-auto-presenter/laravel-auto-presenter
51	PHPOffice/PHPWord
52	atom/atom

53	bobthecow/mustache.php
54	sebastianbergmann/phpunit
55	JosephLenton/PHP-Error
56	dylanfprice/angular-gm
57	proengsoft/laravel-jsvalidation
58	esvit/ng-table
59	christianvuerings/jquery-lifestream
60	assisrafael/angular-input-masks
61	angular/code.angularjs.org
62	thomaswelton/laravel
63	devbridge/jQuery-Autocomplete
64	dyve/jquery-autocomplete
65	domnikl/DesignPatternsPHP
66	plentz/jquery-maskmoney
67	BrentNoorda/toy-piano
68	osteele/functional-javascript
69	pattern-lab/patternlab-php
70	formstamp/formstamp
71	2fdevs/videogular
72	jamesshore/lets_code_javascript
73	PHP-FFMpeg/PHP-FFMpeg
74	ludo/jquery-treetable
75	timmywil/jquery.panzoom
76	brianhaveri/Underscore.php
77	jirikavi/AngularJS-Toaster
78	twilio/twilio-php
79	php/php-langspec
80	elastic/elasticsearch-php
81	kristijanhusak/laravel-form-builder
82	i18next/ng-i18next
83	bitovi/jquerypp
84	twilson63/ngUpload
85	PHPMailer/PHPMailer
86	FreeCodeCamp/FreeCodeCamp

87	react-toolbox/react-toolbox
88	ChenYilong/ParseSourceCodeStudy
89	dennybritz/neal-react
90	yolkjs/yolk
91	herrbischoff/awesome-osx-command-line
92	andybarry/flight
93	AliSoftware/SwiftGen
94	CocoaPods/CocoaPods
95	CocoaPods/Xcodeproj
96	CocoaPods/Specs
97	CocoaPods/cocoapods-plugins
100	Homebrew/homebrew
101	AliSoftware/OHHTTPStubs
102	AliSoftware/OHAttributedLabel
103	czechboy0/XcodeServerSDK
104	Moya/Moya
106	ushahidi/Ushahidi_Web
107	ushahidi/SMSSync
108	Raizlabs/BonMot
109	Raizlabs/RZUtils
110	pinterest/PINRemoteImage
111	lazerwalker/Theseus
112	lazerwalker/cortado
113	lazerwalker/clojurescript-koans
114	intentkit/IntentKit
115	phalt/pokeapi
116	i3/i3status
117	git-up/GitUp
118	DHowett/go-plist
119	Imperiopolis/PHYKit
120	gree/lwf
121	Yubico/python-u2flib-server
122	Chicago/food-inspections-evaluation
123	cernopendata/opendata.cern.ch

124	department-of-veterans-affairs/gi-bill-comparison-tool
125	sorich87/bootstrap-tour
126	rapidpro/rapidpro
127	praekelt/vumi
128	pluspeople/pesaPi
129	nyaruka/smartmin
130	onaio/onadata
131	universalcore/elastic-git
132	frontlinesms/frontlinesms2
133	CodeForAfrica/GotToVote
134	OpenInstitute/OpenDuka
135	eyedol/kasahorow-Keyboard-For-Android
136	rootio/rootio_web
137	chisimba/chisimba
138	buunguyen/octotree
139	github-linker/chrome-extension
140	jasonlong/isometric-contributions
141	muan/github-gmail
142	thieman/github-selfies
143	sindresorhus/github-notifier
144	Justineo/github-hovercard
145	sanemat/do-not-merge-wip-for-github
146	thecodejunkie/github.expandinizr
147	ProLoser/Github-Omnibox
148	Yatser/prettypullrequests
149	ryanflorence/github-plusone-extension
150	benbalter/github-mention-highlighter
151	mike-north/chrome-github-boxcutter
152	libgit2/libgit2
153	JamesNK/Newtonsoft.Json
154	reactiveui/ReactiveUI
155	msysgit/msysgit
156	cefsharp/CefSharp
157	dahlbyk/posh-git

158	NLog/NLog
159	akavache/Akavache
160	Moq/moq4
161	libgit2/libgit2sharp
162	xunit/xunit
163	icsharpcode/SharpDevelop
164	octokit/octokit.net
165	Caliburn-Micro/Caliburn.Micro
166	Reactive-Extensions/Rx.NET
167	nsubstitute/NSubstitute
168	paulcbetts/splat
169	icsharpcode/AvalonEdit
170	facebook/osquery
171	rapid7/metasploit-framework
172	facebook/infer
173	presidentbeef/brakeman
174	jipegit/OSXAuditor
175	radare/radare2
176	beefproject/beef
177	cuckoobox/cuckoo
178	aol/moloch
179	bro/bro
180	jeffbryner/MozDef
181	Netflix/Scumblr
182	google/grr
183	etsy/MIDAS
184	ossec/ossec-hids
185	threatstream/mhn
186	mozilla/mig
187	gamelinux/passivedns
188	sleuthkit/sleuthkit
189	AlienVault-Labs/AlienVaultLabs
190	github/hoosegow
191	adobe-fonts/source-code-pro

192	google/fonts
193	google/roboto
194	mozilla/Fira
195	adobe-fonts/source-sans-pro
196	adobe-fonts/source-serif-pro
197	theleagueof/league-gothic
198	larsenwork/Gidole
199	klepas/open-baskerville
200	scikit-learn/scikit-learn
201	PredictionIO/PredictionIO
202	BVLC/caffe
203	harthur/brain
204	clips/pattern
205	numenta/nupic
206	karpathy/convnetjs
207	ryanb/ruby-warrior
208	JohnLangford/vowpal_wabbit
209	sjwhitworth/golearn
210	johnmyleswhite/ML_for_Hackers
211	h2oai/h2o-2
212	dmlc/xgboost
213	cloudera/oryx
214	shogun-toolbox/shogun
215	dmlc/mxnet
216	nikolaypavlov/MLPNeuralNet
217	mikeizbicki/HLearn
218	apache/mahout
219	SeldonIO/seldon-server
220	jubatus/jubatus
221	danielsdeleo/Decider
222	petdance/ack2
223	mhagger/git-imerge
224	rtyley/bfg-repo-cleaner
225	skywinder/github-changelog-generator

226	eddiezane/lunchy
227	bhollis/jsonview
228	aanand/git-up
230	ShareX/ShareX
231	defunkt/dotjs
232	ggreer/the_silver_searcher
233	robbyrussell/oh-my-zsh
234	gabrielecirulli/2048
235	mozilla/BrowserQuest
236	AlexNisnevich/untrusted
237	doublespeakgames/adarkroom
238	ellisonleao/clumsy-bird
239	Hextris/hextris
247	elixir-lang/elixir
248	scala/scala
250	rust-lang/rust
251	golang/go
252	ruby/ruby
253	php/php-src
254	JuliaLang/julia
255	clojure/clojure
256	erlang/otp
257	jashkenas/coffeescript
258	nim-lang/Nim
259	JetBrains/kotlin
260	elm-lang/elm-compiler
261	timburks/nu
262	gkz/LiveScript
263	Frege/frege
264	groovy/groovy-core
265	stevedekorte/io
266	racket/racket
267	D-Programming-Language/dmd
268	fsharp/fsharp

269	eholk/harlan
270	amber-smalltalk/amber
272	dylan-lang/opendylan
273	chapel-lang/chapel
274	jscs-dev/node-jscs
275	bbatsov/rubocop
276	photonstorm/phaser
277	pixijs/pixi.js
278	BabylonJS/Babylon.js
279	wellcaffeinated/PhysicsJS
280	craftyjs/Crafty
281	cocos2d/cocos2d-html5
282	playcanvas/engine
283	melonjs/melonJS
284	shakiba/stage.js
285	gamelab/kiwi.js
286	ekelokorpi/panda.js-engine
287	jshint/jshint
288	CSSLint/csslint
289	nathanmarz/storm
290	jfeinstein10/SlidingMenu
291	mame/quine-relay
292	CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers
293	wakaleo/game-of-life
294	django/django
295	twbs/bootstrap
296	propublica/upton
297	ocombe/ocLazyLoad
298	powmedia/backbone-forms
299	marcuswestin/WebViewJavascriptBridge
300	trentrichardson/jQuery-Timepicker-Addon
301	phstc/jquery-dateFormat
302	daffl/jquery.dform

303	superfeedr/indexeddb-backbonejs-adapter
304	codenitive/laravel-oneauth
305	anandkunal/ToroPHP
306	ericelliott/essential-javascript-links
307	posabsolute/jQuery-Validation-Engine
308	atom/language-javascript
309	videlalvaro/php-amqplib
310	jrief/django-angular
311	nikic/PHP-Parser
312	FriendsOfPHP/PHP-CS-Fixer
313	rochal/jQuery-slimScroll
314	angular-ui/angular-google-maps
315	maximebf/php-debugbar
316	rackspace/php-opencloud
317	jakesgordon/javascript-state-machine
318	laravelio/laravel.io
319	erikd/language-javascript
320	subtleGradient/javascript.tmbundle
321	FrozenNode/Laravel-Administrator
322	phpservermon/phpservermon
323	rdash/rdash-angular
324	kakserpom/phpdaemon
325	zofe/rapyd-laravel
326	mgechev/angularjs-in-patterns
327	nfriedly/Javascript-Flash-Cookies
328	basco-johnkevin/laravelsnippets
329	jtrussell/angular-snap.js
330	sayanee/angularjs-pdf
331	igorescobar/jQuery-Mask-Plugin
332	bestmomo/laravel5-example
333	phpseclib/phpseclib
334	Halleck45/PhpMetrics
335	sebastianbergmann/phploc
336	davejamesmiller/laravel-breadcrumbs

337	JeffreyWay/Laravel-4-Generators
338	barryvdh/laravel-debugbar
339	caouecs/Laravel-lang
340	Qihoo360/phptrace
341	Maatwebsite/Laravel-Excel
342	banago/PHPloy
343	lingqingmeng/reaction
344	tuupola/jquery_jeditable
345	Jasig/phpCAS
346	rmm5t/jquery-timeago
347	jquery/jquery
348	jquerytools/jquerytools
349	FineLinePrototyping/angularjs-rails-resource
350	McPants/jquery.shapeshift
351	facebookarchive/phpsh
352	maxogden/javascript-for-cats
353	rstacruz/jquery.transit
354	jquery-boilerplate/jquery-boilerplate
355	mixi-inc/JavaScriptTraining
356	paypal/JavaScriptButtons
357	phpbrew/phpbrew
358	mrdoob/three.js
359	carhartl/jquery-cookie
360	jtobey/javascript-bignum
361	PHPIDS/PHPIDS
362	kerphi/phpfreechat
363	rails/jquery-ujs
364	smartiniOnGitHub/grails-angularjs-resources
365	BorisMoore/jquery-tmpl

APPENDIX C: TMPTBFILLING STORED PROCEDURE

```
SET TERM ^ ;

create or alter procedure TMPTBFILLING
returns (LBID integer)
as
declare variable LBID1 integer;
begin
 /*Bugs labels*/
FOR select distinct lbid from labels where
    ((lower(lbnm) like  '%bug' or lower(lbnm) like  'bug%'
       or lbnm containing  'defect'
      or lbnm containing  'security' or lbnm containing
'problem'   )
     and  (lbnm containing  'note' or (not lbnm containing
'not')))  into :LBID1  do
    begin
    INSERT into TMPTLabels (TLGID ,TLBNM, TLBID ) values
    (GEN_ID(GEN_TMPTLabels,1),'B', :LBID1);
    end

        /*Features labels*/
FOR select distinct lbid from labels where
   (lbnm containing  'feature' or lbnm
    containing 'enhancement' or lbnm  containing 'want'
   or lbnm  containing 'task'  )and
      not (lbnm  containing 'want' or (lbnm  containing
'contributor') )
    into :LBID1  do
  begin
    INSERT into TMPTLabels (TLGID ,TLBNM, TLBID ) values
    (GEN_ID(GEN_TMPTLabels,1),'F', :LBID1);
  end

               /*Documents or support labels*/
FOR select distinct lbid from labels where
   lbnm containing  'question' or  lbnm containing  'help'
     or   lbnm containing 'support'or   (lbnm containing
'docs')
       or  (lbnm  containing  'documentation')  or  (lbnm
containing 'faq')
   or (lower(lbnm) like 'doc')
```

```
42      into :LBID1  do
43       begin
44              INSERT into TMPTLabels (TLGID ,TLBNM, TLBID )
45  values
46           (GEN_ID(GEN_TMPTLabels,1),'D', :LBID1);
47       end
48
49          /*wontfix labels*/
50  FOR select distinct lbid from labels where
51       lbnm containing  'wont' or lbnm containing  'won''t'
52       or (lbnm containing  'no' and lbnm containing  'fix')
53         or (lbnm containing  'partial' and lbnm containing
54  'fix')
55  into :LBID1  do
56       begin
57        INSERT into TMPTLabels (TLGID ,TLBNM, TLBID ) values
58       (GEN_ID(GEN_TMPTLabels,1),'W', :LBID1);
59       end
60
61          /*duplicate labels*/
62  FOR select distinct lbid from labels where
63       lbnm containing  'duplicate'
64       into :LBID1  do
65       begin
66              INSERT into TMPTLabels (TLGID ,TLBNM, TLBID )
    values
           (GEN_ID(GEN_TMPTLabels,1),'2', :LBID1);
       end
    end^

    SET TERM ; ^
```

APPENDIX D: EVENTPARSER STORED PROCEDURE

```
  Create procedure eventparser  (PRJID integer)
 1 RETURNS  (isTp  char(1),LGTH  float,    RPTM  integer,    n_comm
 2 integer, Assgn char(1),
 3      wontfix char(1),dupl char(1),PPart integer, respTm float,
 4 firstLabel float)
 5
 6  as
 7  declare variable labelnm integer;
 8  begin
 9    isTp ='f';
10    LGTH =1.2;
11    RPTM = 2;
12    n_comm =3;
13    Assgn ='t';
14     wontfix ='t';
15     dupl ='f';
16     PPart =4;
17     respTm =1.4;
18     firstLabel =3.3;
19
    end^
```

APPENDIX E: COMMITSPERUSER STORED PROCEDURE

```
  create or alter procedure COMMITSPERUSER (
 1     PRJID integer)
 2 returns (
 3     NUMBEROFCOMMITS integer)
 4 as
 5 declare variable CURRENTUSERID integer;
 6 begin
 7 for select distinct userid from commits where PRJID=:prjid
 8 into :currentuserid
 9 do
10 begin
11              select    count    (*)    from    commits    where
12 userid=:currentuserid and prjid=:prjid
13         into :NUMBEROFCOMMITS;
14        /* if (NUMBEROFCOMMITS>5) then */
15     suspend;
16   end
   end^
```